UNIVERSIDADE DE LISBOA
FACULDADE DE CIÊNCIAS
DEPARTAMENTO DE INFORMÁTICA

CORQ - Code Review for Quality Measurement

Alexandre Miguel Costa Ramos

Mestrado em Informática

Trabalho de projeto orientado por:
Prof. Doutor João Pedro Guerreiro Neto

2022

Acknowledgments

This thesis is, just like any other period of time of my life, something that will be a part of me and that has shaped me into the person I am now. It was not an easy journey, but I know for a fact that it has helped me grow as a person and as a professional. And just like any journey, not only the destination is important, but the journey itself is important. This thesis has shown me that my university journey is coming to an end, and that everything that is good in life has an ending to it as well. I will have memories of FCUL for the rest of my life, and that includes every person that has been on this journey with me.

First of all I would like to thank the people that have guided me in this master thesis. Luís Pereira and Ricardo Mascarenhas, the two people that guided me every week for ten months in the development of this project. They made my experience at Opensoft the most welcoming and positive possible. I want to thank Professor João Neto, not only for the times he teached me at FCUL as my Professor, but also for guiding me in this master thesis in these ten plus months as well. These three people knew when to tell me it was okay that something was not going right, and that everything had a solution if I just thought in different ways about it, and a solution would always be found.

I would like to thank every single friend that I have made in this journey, since every single one of them has a special meaning to me, and every memory I have had with them, will be with me for the next years I have to come. They helped me go through the rough times, and always shown to be there when I needed them the most.

I want to thank my girlfriend, for every second she has been in my life, may that second be good or bad, she is always there for me no matter what, and while I was working on this thesis, it was no exception.

And finally, I would like to thank all my family, for making me be the person that I am right now, and for motivating me everyday of my life.

The acknowledgements would be too long if I had to thank my friends, girlfriend and family for everything they do for me every day.

Thank you all for everything.

To my family, girlfriend and friends.

Resumo

Este relatório foi desenvolvido com base no projeto realizado na empresa de *software* portuguesa Opensoft no âmbito do meu Mestrado em Informática de 6 de setembro de 2021 a 30 de junho de 2022.

Qualidade e produtividade são alguns dos, senão os mais importantes fatores em qualquer empresa de *software* na realidade atual para o seu crescimento e funcionamento.

Portanto, há a necessidade de obter métricas de qualidade e produtividade sobre o trabalho realizado pelos desenvolvedores de *software*.

Code Review for Quality Measurement (que irá ser referenciado como CORQ) é uma ferramenta desenvolvida de raiz por uma pequena equipa da Opensoft incluindo eu, empresa esta onde esta ferramenta foi desenvolvida. A ferramenta foi desenvolvida para efeitos desta tese de mestrado. A ferramenta fornece métricas sobre a produtividade e qualidade de *software* dos colaboradores da Opensoft, projetos da Opensoft ou da própria empresa no geral.

A ferramenta pode ser utilizada em qualquer empresa que utilize as mesmas plataformas para obter as métricas.

Este problema pode ser resolvido com trabalhos futuros, caso queiramos expandir a ferramenta para ser utilizável em outras empresas de *software*.

O CORQ tenciona utilizar múltiplas interfaces externas como o *SonarQube*, repositórios git que se encontram no *Bitbucket*, *Jira*, *Confluence* e repositórios SVN para obter informações sobre a Opensoft e assim calcular métricas sobre a produtividade e qualidade dos seus colaboradores e do *software* produzido por eles.

Com o uso do CORQ espera-se que os colaboradores da empresa fiquem mais motivados, aumentando a sua produtividade, economizando recursos humanos e economizando tempo de análise do *software* produzido. O CORQ pode ser usado complementarmente à revisão de código e outras técnicas e *software* de garantia de qualidade de *software* utilizadas na Opensoft.

A análise manual de código não será substituída devido aos benefícios que acarreta tal como o conhecimento ganho, a prevenção de acidentes, segurança, acompanhamento da evolução de um projeto e manter padrões no código, no entanto, usar o CORQ economiza tempo aos revisores de código, aumentando assim a produtividade de todos os colaboradores da Opensoft.

Todo o trabalho que for descrito neste resumo foi desenvolvido por mim, salvo exceções que serão explicitamente explicadas, visto que trabalhei numa equipa onde eu era o único desenvolvedor de , com a ajuda e orientação do meu supervisor Luís Pereira e do arquiteto de *software*, o Ricardo Mascarenhas.

O CORQ tem como objetivo facilitar a deteção de problemas, sendo que todos os dados produzidos pelo CORQ serão analisados por todos os utilizadores da Opensoft, tais como os diretores de projeto, gestor de projeto, e desenvolvedores de *software*.

O projeto foi dividido em algumas fases para o planeamento tais como a análise e desenho, a implementação, os testes, a distribuição e por fim a escrita desta tese de mestrado. As primeiras secções descritas aconteceram pela ordem descrita, tendo em conta que a análise e desenho teve um planeamento de seis semanas, a implementação um planeamento de onze semanas, a distribuição um planeamento de duas semanas, e os testes que acontecem nas últimas cinco semanas de implementação.

Em algum trabalho relacionado com estes tópicos, Belachew *et al.* [1] discute a importância e complexidade da análise de *software* ao utilizar métricas de *software*. Eles também discutem como, uma vez que o padrão de qualidade pode diferir de uma organização para outra, é melhor aplicar as métricas de *software* para medir a qualidade do *software* e as ferramentas de métricas de *software* mais comuns para reduzir a subjetividade das falhas durante a avaliação da qualidade do *software*, que se aplica ao CORQ. Para os autores, citações que referem tais como "If you aren't measuring, you aren't managing" [2] são importantes para introduzir o tema e a importância de medir métricas de *software*. Mladenova [3] também refere esta mesma citação. A autora também discute como os testes de qualidade de *software*, manuais ou automáticos, por vezes não são suficientes para dar uma visão mais profunda e completa de todo o processo de desenvolvimento de *software*, e isso pode ser alcançado selecionando adequadamente as métricas de *software* que podem ser usadas para a avaliação de risco adequada e avaliação do desenvolvimento. Por fim, a autora refere como ferramentas que podem ir ao encontro dos propósitos de garantia de qualidade de *software* referenciadas podem ser úteis para organizações que desejam acompanhar o seu desenvolvimento, custos, riscos e defeitos, e é esse um dos objetivos do que o CORQ pensa concretizar. Moser *et al.* também discute alguns pontos que se relacionam com o CORQ tal como referem no seu estudo como métricas de processo mostram dar informação mais valiosa ao ser analisada do que métricas de código, sendo que o CORQ tem como ideia de métricas importantes a implementar métricas relacionadas com a documentação e gerenciamento de tempo tal como o *Confluence* e *Jira*.

Em termos de *state-of-the-art*, um exemplo semelhante foi desenvolvido como Thiruvathukal *et al.* [4] projetaram, e implementaram um protótipo de uma plataforma similar, que se focava em métricas de projetos no *GitHub*, mostrando métricas no *GitHub* sobre a equipa trabalhando nesses projetos e a saúde dos projetos em vez de métricas sobre uma pessoa singular tal como o *GitHub* mostra por predefinição.

Após a análise do problema, o desenho da solução é essencial para que a implementação seja o mais efetiva e produtiva possível. Ideias iniciais do desenho podem ser mudadas ao longo da implementação e foi o que aconteceu. Estes componentes existem numa máquina de produção.

Apesar de a implementação da solução ter como visão e desenho de utilizar todas as plataformas utilizadas pela Opensoft para obter indicadores e métricas de quantidade e qualidade, o tempo disponível somente foi suficiente para implementar métricas quantitativas retiradas a partir da API do *Bitbucket*. No entanto, a implementação para acesso *APIs* e exploração de qual informação é possível retirar a partir das outras plataformas tais como *Jira*, *Confluence* e *SonarQube* foi feita, sendo que tem como base o *Retrofit2* e *OkHttp3*, *frameworks* utilizadas para acesso a *APIs* externas a partir de pedidos *REST*. O CORQ pode ser utilizado tanto na máquina servidor ou pelos colaboradores diretamente nas suas máquinas. Vários *Jobs Batch* foram implementados com diferentes fins, tais como *Jobs* para clonar repositórios para a máquina local, gerar métricas, dar *reset* no *index* do *ELK Stack*, para inserir projetos não existentes no *Bitbucket* para a base de dados do CORQ, um *Job* para popular a base de dados com projetos existentes no *Bitbucket* e usuários dos colaboradores que existem no *Crowd*, e um *Job* para enviar dados sobre métricas de um certo intervalo de tempo por *e-mail*. O *ELK Stack* foi configurado para funcionar com o CORQ a partir de ficheiros *log* que são gerados com o CORQ quando novas métricas são geradas. Os dados dentro deste ficheiro log são encaminhadas a partir do *Filebeats* para o *ELK Stack* na máquina servidor e mostrados a partir da visualização que o *Kibana* fornece. A instalação dos módulos *Logstash*, *Kibana* e *Elasticsearch* na máquina servidor e configuração do projeto no *Jenkins* foram concretizados pelo meu supervisor Luís Pereira. Esta visualização é altamente configurável e intuitiva. Para finalizar a implementação, alguns testes foram implementados para as classes de entidades e repositórios, tais como nas classes de serviço de ambos os módulos implementados, sendo que nem todos os testes pensados foram implementados devido à falta de tempo disponível para a implementação dos mesmos.

Apesar de o planeamento não ter sido cumprido totalmente, a implementação desta solução nos dez meses que fiz parte desta experiência mostrou-se uma grande realização tanto pessoalmente como profissionalmente, e a solução mostrou ter feedback positivo dos colaboradores na Opensoft. A visualização dos dados gerados pelo CORQ Batch mostraram-se altamente configuráveis, sendo fácil retirar informação valiosa para a empresa e os seus colaboradores a partir dos dados e gráficos gerados a partir da solução como um inteiro.

Apesar das dificuldades sentidas no decorrer deste projeto, o principal objetivo aquando implementando o projeto foi sempre ter uma solução com uma implementação bastante extensível e sólida para que a adição de qualquer métrica que quisermos adicionar à solução seja fácil, e ter uma solução utilizável pelos *stakeholders* definidos, para eles

conseguirem obter informação valiosa a partir do uso da mesma, e a equipa que trabalhou e guiou este projeto, e a Opensoft sentiram que isso foi cumprido.

Poderão ser adicionadas outras *features* que não foram pensadas pela equipa que desenvolveu o CORQ, sendo que a extensibilidade do CORQ dá-me confiança suficiente para dizer que qualquer nova *feature* pensado para ser implementado no CORQ pode ser implementado sem esforço excessivo.

Palavras-chave: qualidade, produtividade, métricas, revisão de código, medição

Abstract

This report was developed based on the project held in the portuguese software company Opensoft in the context of my Master's degree in Computer Science from September 6th 2021 to June 30th 2022.

Quality and productivity are some of, if not the most important factors in any software company in today's reality for its growth and operation.

Therefore, there is a need to obtain quality and productivity metrics for the work conducted by software developers.

Code Review for Quality Measurement (as will be referenced from now on as CORQ) is a tool developed from scratch by a small team in Opensoft including me, which is the software company where this tool was developed, for the purpose of this thesis, that provides metrics on the productivity and quality of software of a company's employees, projects, or itself overall.

The tool may only be used in Opensoft initially due to the interfaces that are used to obtain these metrics and the specifications needed to implement the project.

This problem can be solved with future work, in case we want to expand the tool so it is usable in other software companies.

CORQ intends to use multiple interfaces such as SonarQube, Jira, Confluence, Git repositories in Bitbucket and SVN repositories to obtain information about Opensoft's artifacts and thus calculates metrics on the productivity and quality of its employees and the software produced by them.

With the use of CORQ it is expected for the company's employees to be more motivated, increasing their productivity, saving human resources, and saving analysis time spent on the software produced. CORQ can be used complementary to code reviewing and other software quality assurance techniques and software.

Keywords: quality, productivity, metrics, code review, measurement

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This report describes the work done in a project called Code Review for Quality Measurement, at Opensoft by me. Opensoft is a Portuguese software company that is focused on developing solutions to ease people's daily basis. With the growth of Opensoft's scale due to the increasing need for IT solutions in all essential services to today's society, Opensoft has to increase its productivity and the software quality of its employees. This document presents the author's efforts regarding this project. Any exception will be noticed in the text.

This growth in Opensoft's scale makes obtaining information about the productivity and quality of the software of its employees a task that consumes a lot of time and resources, and despite the existence of tools that give us information after the static analysis of the software, there is a need for an automatic tool that provides us this information per employee.

CORQ (Code Review for Quality Measurement) is a tool that allows us to do just that. CORQ uses information from different sources that Opensoft uses and obtains new indicators on productivity and software quality for each employee in the company.

By using CORQ we save time and human resources that would otherwise have been needed to do CORQ's work manually. Doing this work manually for the productivity metrics (lines added, lines changed, and lines removed) would take more time than what CORQ does automatically. These values are discussed in this document in further depth in Chapter 3.

1.1 Context

Most of Opensoft's software solutions are solutions used daily to ease a person's everyday life, such as tax calculation, complex forms, and notary work, among other types of solutions. These applications handle a lot of people's sensitive information such as ID and financial information. Opensoft's applications must be designed and implemented in the best way possible. But what does best mean in software?

Quality in software can be subjective. Different companies and applications may prioritize or find more important specific quality metrics. Code for big applications developed by companies such as Opensoft can be very complex. With the rise in size and consequently complexity of code, these applications can be hard to review on a daily basis, to ensure the quality wanted and needed by these companies.

Quality metrics important for complex applications can translate to less productivity in the company. Quality metrics such as Maintainability, Reliability, and Security are good examples. Software maintainability is defined as the degree to which an application is understood, repaired, or enhanced.

Software Reliability is the probability of failure-free software operation for a specified period of time in a specified environment. Software security is an idea implemented to protect software against malicious attacks and other hacker risks so that the software continues to function correctly under such potential risks. These three examples of quality metrics translate to problems that exist, and that should be taken into account when developing new software. They should always be at their best, and to keep them at their best, they must be reviewed constantly.

1.2 Problem

As the number of collaborators working at Opensoft increases, there is more and more code produced every day. Code at Opensoft is reviewed daily from a quality and productivity point of view. Although some platforms used by the company such as Bitbucket and SonarQube can ease this process, it is still a time-consuming task.

Bitbucket can give us information on lines added and lines removed on a date range for each collaborator though it bears some limitations, for example, lines added and removed for the whole project or one single collaborator on all projects they are a part of and contributed towards in a specific period.

SonarQube can give us information on the quality of code of a specific project or branch of that project but does not aggregate that information in terms of collaborator. Information such as how a collaborator's quality metric is on a project or information about all the projects they worked on aggregated can be analyzed and help detect quality issues.

For example, if in the last week a collaborator has had a bad maintainability rate towards a specific project, but their overall maintainability rate is positive, there might be an issue with that collaborator in that specific project. Maybe that collaborator might not be working well with that project team. Perhaps that collaborator is not very interested in that project's theme. And that is the reason we need information aggregated and observable in different ways.

This is a problem with the current tools. They do not aggregate information about the collaborators in all the ways we need them to, in order to detect all the metrics in a dynamic and observable way, and fast.

On average, obtaining information about quality and quantity metrics, having into consideration different ways of aggregating these metrics, would take one hour per repository. This information would still be not aggregated, since it is visualized as raw data, which can be very confusing and not very intuitive. With this information aggregated, it is easily comparable from each repository to another, and easily visualized and analyzed to conclude results much easier.

1.3 Motivation

Code quality is most important for Opensoft's business model. The company is always looking to produce more and better, and currently, there are tools for static analysis of code quality, for example, SonarQube, which is being used in Opensoft's projects.

This tool analyzes the quality of the software produced by aggregating this information by project. However, it does not allow you to view quality information about the code produced by each employee/developer overall, or in specific projects.

With the current size of the company as well as the expected growth, it is not sustainable to only have other employees assess the quality of code produced by colleagues, and therefore it was thought necessary to invest in an automatic solution such as CORQ. SonarQube will be one of the sources of information that is used in this individual quality measuring tool.

But we have others, for example, the collaborative tools to support software development at Opensoft, such as Git repositories hosted in Bitbucket and SVN repositories, Confluence, and Jira by Atlassian.

1.4 Goals

To ease the detection of issues, CORQ will be incorporated into Opensoft as a complementary tool to the work being done at the moment, being used by Project Managers and Project Directors by analyzing the tool's analysis report, so they can detect problems and situations in which they can intervene in the way they deem most appropriate, to motivate employees of Opensoft. CORQ will make the detection of issues related to software produced or collaborators, by analyzing the quality and quantity of code produced every day, week, or year much easier and better. Some problems may not always be obvious to understand, such as a collaborator that produces code that only fixes other colleagues' bugs or tests, or some issue about a specific project that can lead to low productivity in it.

Employees need to be motivated and work on something that makes sense to them, and that is an issue that software companies such as Opensoft struggle to detect and fix.

Developers will use CORQ also to see information on quality and productivity indicators, however, they also are the most active stakeholder in CORQ since they provide the information to generate these indicators.

1.5 Methodology

CORQ was developed using Opensoft's methodology. All the work done at the company to develop the application followed a professional approach. Opensoft's methodology is based on standards recognized by the market, following an Agile approach, CMMI practices, and ITIL framework for TI service management. This methodology accelerates the process of development, grants better results and deadline predictability, and product quality.

1.5.1 Planning

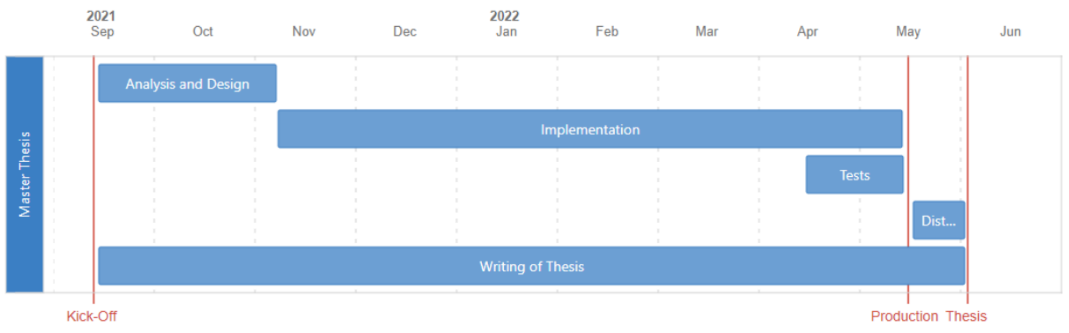


Figure 1.1: Project Timeline

In the Project Timeline in Figure 1.1, we have the following sections in this order: Analysis and Design, Implementation, Tests, Deployment. There is also the writing of this thesis that is done throughout the project. We also have some checkpoints, which are the following: Kick-off, Going into Production, and Thesis Delivery.

In Analysis and Design, several CORQ planning and architecture documents were produced, such as the vision document, project plan, and architecture document (SAD).

In Implementation, the SRS was developed (User stories), and the implementation of the whole application.

In Tests, the configuration of the pilot ambient (PRD), and fixing its issues.

In Deployment the application enters production, fixing its issues and there was also the elaboration of installation and usage guides.

In Writing of Thesis, this thesis was planned to be written at the same time as there was work being done in the application.

Some changes were made to the plan, such as the time allocated to writing this document throughout the whole process instead of allocating that time to the last four weeks. These four weeks were allocated to Implementation.

Another change to the plan was to work an extra month for the implementation of the project since the time that was planned for nine months was not enough. During this extra month, CORQ was successfully deployed, tested, and evaluated in a production environment.

1.5.2 Analysis and Design

Analysis and design of the application. This section describes the initial approach to the application and organization of how the problem will be tackled. In any project, good analysis and design can help reduce unnecessary work, making the implementation more organized, and straightforward. This section has an expected time of conclusion of six weeks.

Vision Document

Guidelines of the system to be developed in terms of needs, requirements, restrictions, and general description of the solution, its functioning, main characteristics, and implementation priority. This document is made at the project start and it is not changed afterward.

Specification of Functional Requirements - SRS

Defines and details the functional requirements identified in the vision document, describing them from the point of view of the entities that will use them and referring to the interfaces they may have with other systems. The first version of this document is made at the project start and whenever there are any changes to it.

Software Architecture Document - SAD

It defines the system's architecture taking into account the context in which the system to be developed is inserted, which are the objectives and constraints. This document defines the infrastructure, components, and interfaces, and specifies the integration between components and other software modules. The first version of this document is made at the project start and whenever there are any changes to it.

Tests Plan

Test plan definition document through the definition of test requirements (functional, integration, performance, load, security, etc.), details the necessary conditions for the test environment (hardware, software, and tools), and the main milestones to be achieved. The first version of this document is made at the project start and whenever there are any changes to it.

1.5.3 Implementation

There were three sections of implementation planned and this section had an expected time of conclusion of twenty-seven weeks. The first one is the core of the application, which had an expected implementation duration of eleven weeks, and includes:

- Definition of user stories (SRS)
- Definition of a productivity indicator
- Definition of a quality indicator
- Test plan definition
- Implementation of the automatic unit and end-to-end tests
- Obtaining one productivity indicator
- Obtaining one quality indicator
- Consultation and search of indicators by an employee
- User authentication
- Functional and application tests
- Development Environment Configuration (DEV)
- Setting up the version building process

Following the priority that has been defined in the application, the second section, which had an expected implementation duration of three weeks, includes:

- Definition of user stories (SRS)
- Query aggregated by indicator
- General aggregated query
- Functional and application tests

The third section is iterative, in which each iteration is based on adding a new indicator. Each iteration, which had an expected duration of implementation of 2 weeks, includes:

- Define the indicators to implement
- Implementation of the automatic unit and end-to-end tests
- Updating user-stories
- Obtaining new productivity and quality indicators
- Functional and application tests

At the same time as implementation, tests were also being developed. The test section includes a pilot environment setup (PRD) and pilot version installation. This section had an expected time of conclusion of five weeks.

After implementation we have the following sections:

- **Deployment** - This section includes the entry of CORQ into production, further ahead the monitoring and resolution of anomalous situations in production, and the elaboration of the installation and usage guide. This section had an expected time of conclusion of two weeks.
- **Thesis Writing and Delivery** - This thesis was written at the same time as other sections of the project planning, from start to finish.

1.6 Structure of the document

This document is organized as follows:

- Chapter 1 - Introduction - Introduces the context, problem, motivation, and goal for this thesis.
- Chapter 2 - Technologies and Related work - Describes technologies used in the application and related work.
- Chapter 3 - Analysis - Describes the analysis made on the problem before tackling it, and solutions.
- Chapter 4 - Design - Describes the design of the application.
- Chapter 5 - Implementation - Describes the implementation of the application.
- Chapter 6 - Conclusion - Describes the conclusion about the evaluation and work done on the solution, and future work to be made on the application to complement the existing work.

Chapter 2

Technologies and Related work

2.1 Technologies

In this section, some technologies and tools that were used to develop the application will be described.

2.1.1 At the workplace

Since the project was developed at Opensoft, I had the experience to work with the same methodologies as the workers at Opensoft. Opensoft applies several platforms and concepts to improve its organization and increase its productivity. In this subsection, these platforms and concepts will be described, since they were used on a daily basis.

Communication

Communication using Gmail and Google Hangouts platforms is performed by e-mail, direct messages, voice, or video call. This helped the development of CORQ, including twice-a-week meetings with the supervisor and a software architect.

Confluence

Opensoft is based on tools developed by Atlassian [5]. The tools developed by Atlassian are easy to use and work well together for the purpose of a company's project and collaborator management, and tools developed by Atlassian have shown to meet requirements that are considered standard in software companies for management and traceability of collaborators' everyday work, due to easy to configure and to implement third-party plugins. It makes these tools customizable to every software company's different needs and requirements.

Confluence is a web-based corporate wiki (collaboration software) developed by Atlassian. Confluence works as a wiki or a forum, where any important information can be published. Confluence was used to document everything related to CORQ while it was being developed.

Jira

Jira is another tool developed by Atlassian. It is a tool that allows the monitoring of tasks and tracking of single projects in managing all your activities from the place. Jira was used to plan and register every hour worked on the project. Good time management is very important to any project implementation, and that applies to CORQ.

Crowd

Crowd is the centralized SSO tool used by Atlassian for Jira, Confluence, and Bitbucket. It is used in CORQ to gather information about the collaborators. Crowd was also used to login into all tools owned by Atlassian and used by collaborators at the company.

Bitbucket Repositories

Bitbucket is a Git-based source code repository hosting service owned by Atlassian. Bitbucket is used to host Opensoft's projects.

Apache Subversion (SVN) Repositories

Apache Subversion (often abbreviated SVN, after its command name svn) is a software versioning and revision control system distributed as open source under the Apache License.

Software developers use Subversion to maintain current and historical versions of files such as source code, web pages, and documentation. Opensoft also uses this kind of repository.

SonarQube

SonarQube (formerly Sonar) is an open-source tool developed by SonarSource for continuous inspection of code quality to perform automatic reviews with static analysis of code to detect bugs, code smells in 17 programming languages.

SonarQube offers reports on duplicated code, coding standards, unit tests, code coverage, code complexity, comments, bugs, and security recommendations.

SonarQube is used on a daily basis at Opensoft to improve the code quality at the company.

Jenkins

Jenkins is an open-source automation server. It helps automate the parts of software development related to building, testing, and deploying, facilitating continuous integration and continuous delivery. It is a server-based system that runs in servlet containers such as Apache Tomcat. It is shown that the design and implementation of a CI/CD schema based on Jenkins and Ansible saves resources, as the integration and deployment of projects are accomplished automatically whenever there's an update to a repository, making it so that project members can focus on other tasks at hand [6]. It is the main tool for continuous integration and deployment used at Opensoft.

2.1.2 Project specific

Implementation tools and technologies used by me in developing this project will be described. They include technologies and tools used at Opensoft and some that had to be thoroughly thought out before being used for the implementation of the project.

Java

Java is a high-level, class-based, object-oriented programming language that is designed to have as few implementation dependencies as possible. It is a general-purpose programming language intended to let programmers write once, and run anywhere, meaning that compiled Java code can run on all platforms that support Java without the need to recompile.

In the context of CORQ's development, Java 11 was used throughout the whole development. Java is the programming language used at Opensoft, with the use of frameworks that ease development in Java.

Apache Maven

Maven is a build automation tool used primarily for Java projects. Maven can also be used to build and manage projects written in C#, Ruby, Scala, and other programming languages.

This is also one of the pillars behind the development of CORQ and projects at Opensoft, besides Java.

Spring

Spring is an open-source application framework of a control container for Java. Although the framework does not impose any specific programming model, it has become popular in the Java community as an addition to the Enterprise JavaBeans (EJB) model.

Spring features its own model–view–controller (MVC) web application framework. It has various frameworks, each of them being focused on a specific layer of an application.

For CORQ, the extensions of Spring used were Spring Boot, Spring Batch, and other small extensions.

Spring Boot Extension is Spring's convention-over-configuration solution for creating stand-alone, production-grade Spring-based Applications that you can "just run".

Spring Batch is a framework for batch processing that provides reusable functions that are essential in processing large volumes of records. It is used to read, process, and write all records about metrics and measurements in the application.

ELK Stack and Filebeats

The ELK Stack is the combination of three open source products developed by Elastic, them being Elasticsearch, Logstash, and Kibana, that has been complemented by Filebeats in this project. The ELK Stack shows to have better performance and visualization with Kibana than some commercial solutions [7].

Elasticsearch is an open-source, full-text search and analysis engine.

Logstash is a log aggregator that collects data from various input sources, then filters and transforms the data, and sends that filtered and transformed information to Elasticsearch or other output destinations.

Kibana is a visualization layer that works on top of Elasticsearch, providing users with the ability to analyze and visualize the data.

Filebeats are lightweight agents that are installed on edge hosts to collect different types of data for forwarding into the stack.

REST

REST (REpresentational State Transfer) is a communication protocol that emerged intending to simplify access to Web services. The main differences compared to the traditional protocol (SOAP) are the semantic use of HTTP methods (GET, POST, PUT and DELETE), the lightness of data packets transmitted over the network, and the simplicity of using various formats for data representation, such as JSON (one of the most used), XML, RSS, among others, making it unnecessary to create intermediate layers (Ex.: SOAP envelope) to encapsulate the data.

HTTP

HTTP (Hypertext Transfer Protocol) is a standard protocol typically used by browsers to communicate with application servers and obtain resources such as HTML documents. Although developed for communication between browsers and application servers, it can be used for other purposes, including as an application transport layer for higher-level protocols such as REST and SOAP. It follows a classic client-server model.

JDBC

JDBC (Java Database Connectivity) is a software component that allows an application in Java language to communicate with a Database. To connect to the Database, JDBC requires drivers that provide the connection and implement the protocol to transfer the query and the result between the client and the database.

MySQL (MariaDB)

MariaDB is one of the most popular open-source relational database technologies. It is developed by the same developers as MySQL.

This database technology was used while developing CORQ and can be migrated to standard MySQL at any time with ease.

2.2 Related work

CORQ is based on concepts such as code reviewing, quality and quantity indicators, and measurements. These concepts have become more and more important as software evolves.

These concepts exist daily in most if not every software company, and some use software already available in the market such as SonarQube, and add-ons to existing software that are not focused on code reviewing such as Bitbucket, which are platforms used by Opensoft.

Belachew *et al.* [1] discuss the importance and complexity of the analysis of software using software metrics. They also discuss how since the quality standard can differ from one organization to another, it is better to apply the software metrics to measure the quality of software and the current most common software metrics tools to reduce the subjectivity of faults during the assessment of software quality, which applies to CORQ. For Belachew *et al.*, quotes referring to how “If you aren’t measuring, you aren’t managing” [2] are important to introduce the theme and importance of measuring software metrics. Software testing and analysis are costly and use many human resources and money, and measurements on software can be done in any development phase of a project to determine cost estimation and resources needed. To Belachew *et al.*, as the size and complexity of software increases, manual inspection of software becomes a harder task, which makes software defect prediction important, using software metrics to analyze the quality of software shows to help this prediction and save resources, including human resources and money.

Mladenova [3] discusses how software quality tests, manual or automatic, are often not enough to give a more deep and thorough view of the whole process of software development, and that can be achieved by properly selecting software metrics that can be used for proper risk assessment and evaluation of the development. The author analyzes different quality models and software metrics, dividing software metrics into three categories, and each of the categories can be used to retrieve different information needed to improve software quality. Conclusions about why having and using a set of metrics is important are also done, being some arguments on how an end product with more quality results in higher customer satisfaction, how different metrics can be used for monitoring when a process or a service is going out of control, how metrics can improve the development of a product, and give a clearer idea about the state of the development team and its practices, and that metrics can enhance the software maintenance. The author talks about how Software Quality Assurance helps achieve quality in terms of meeting requirements in the product or service developed. Like Belachew, Mladenova talks about the importance of assuring quality at every development phase of projects and mentions the same quote by Campbell [2] as Belachew, showing again how important and necessary the quality assurance of software is. Finally, the author refers to how tools that can meet the purposes of software quality assurance referenced can be useful for organizations that want to track their development, costs, risks, and defects, and that is one of the objectives CORQ is thought out to accomplish.

In a study regarding software defect prediction, Moser *et al.* [8] show that although in some studies process metrics can be more effective for software defect prediction, others say that code metrics are more effective. Moser *et al.* conclude that the results of the research strongly endorse building defect predictors using change data of source code, which can be retrieved easily from code repositories, outperforming predictors based on static code attributes and tools that predict defects using code metrics, meaning process-related metrics, contain more discriminatory and meaningful information about the defect distribution in software than the source code itself. This is an important point that was thought to be implemented on CORQ by having metrics related to Confluence and Jira included in the final product, to not only have metrics about the code itself but the whole process of a project. In a final point of the conclusion, the authors also conclude that if there is the disposal of code metrics, a combined model might produce slightly better results, which shows that using all kinds of metrics from all platforms used at Opensoft might produce better results, than just focusing on product or process metrics.

In terms of state of the art, a similar example was developed as Thiruvathukal *et al.* [4] have designed and implemented a prototype of a similar platform, that focused on metrics of projects in GitHub, showing metrics on GitHub about the team working on those projects and the projects' health instead of metrics about a singular person.

Chapter 3

Analysis

As mentioned before, the growth of any company, and in this case Opensoft, brings some problems. A big one is the growth of the number of new projects the company works on every day. More code and documentation produced means more reviewing time spent on it.

On average, one hour is spent per Git repository in Bitbucket by collaborators reviewing its code and documentation, meaning one-eighth of the time a day (assuming the collaborator reviews one repository a day). It includes reviewing code pull requests at Bitbucket, reviewing the code deployed on SonarQube, and documentation in Jira, Confluence and SVN.

Since Opensoft is based on a team-based organization for their projects, it would be much simpler than a team leader could have the information about the work done by their team collaborators aggregated in the same place just one click away.

The idea behind CORQ is to use the APIs of the platforms used by Opensoft mentioned above to get information about the artifacts produced by collaborators, use the metrics available by the platform's APIs and new ones generated by CORQ, and aggregate this information in one single place.

So if a team leader with four collaborators that work under them on three different projects, they can review the quality metrics of all four collaborators about all three projects in one single place without having to review and evaluate themselves on all platforms that Opensoft uses.

Although a tool like CORQ would not substitute code reviewing or SonarQube as a whole, even though it is a costly task, it has its benefits such as learning, accident prevention, security, tracking the evolution of a project, and maintaining standards in the code, hypothetically, using CORQ saves time for the reviewers, thus increasing the productivity of Opensoft collaborators, but is just a complement to the already existing work.

Apart from the time and resources saved, it can make the review easier and clearer for the reviewer, making it fairer for collaborators being reviewed. It works as a centralized system of metrics analyzed from all platforms that Opensoft uses, and that includes SonarQube, which is a platform that used continuous inspection of code quality to perform automatic reviews with static analysis of code to detect bugs, and code smells.

More fairness in reviews means happier and more satisfied collaborators. A company where every collaborator gets treated fairly and is given credit for their work is a better company, and that is one of the main objectives of CORQ.

3.1 Necessary conditions

Some conditions are necessary for the project to be developed and delivered according to the company's standards as shown in Table 3.1. This is a process needed to get a global view of what we need. This is also important in case we are delivering this to an outside company, so we know exactly what we need to develop for the product.

ID	**Identified needs**
ND01	Obtain and record information on **productivity** indicators by employee.
ND02	Obtain and record information on **quality** indicators by employee.
ND03	Allow the **consultation and exploration** of productivity and quality data by employee, respective indicator values, and time periods.
ND04	Design a solution architecture that allows **extension to new productivity and quality indicators.**
ND05	Only Opensoft users can access this system and therefore it is necessary to guarantee the security of this private information, through **authentication** based on Crowd, which is the authentication server at Opensoft and with **SSO mechanism.**
ND06	Implement a web interface using the best **usability** and usage practices in mobile devices (responsive).
ND07	The system has to **calculate** the necessary information every day, and this calculation cannot exceed 24 hours.
ND08	The **response time** for querying information in this system cannot exceed 1 minute.
ND09	This application is developed in **Java version 11 (minimum)** and users access the system via a modern browser: **Google Chrome or Firefox**.
ND10	The project **documentation** consists of what is defined in the methodology for developing solutions at **Opensoft.**
ND11	For this project it is also necessary to prepare the **documents** required by **FCUL**.

Table 3.1: Necessary conditions

These conditions were the base of what we should have implemented when the application is developed.

3.2 Users

After knowing the conditions that we need for our application, we should analyze which stakeholders and users will use our application, and in what context it will be used. CORQ is meant to be used in a way that eases the everyday life of collaborators at Opensoft, and as such, they are very important to be considered.

User	Stakeholder
Collaborator Project Manager Project Director	Opensoft

Table 3.2: Users

See Table 3.2 for a reference on all the users considered for CORQ. The environment we want users to be able to use CORQ is:

- The user will be able to access it at home or at work from a computer.
- The solution is a web application available through a browser, requiring an internet connection.
- The application usage time for code qualification query will depend on the user's intentions.
- The user will need to use a recent version of Google Chrome.
- The projects to be analyzed by the tool must be connected to Bitbucket from SonarQube.
- The user must have a Crowd account for authentication and access to information.

3.3 Initial global vision

A global vision of how the users described in the section before interact with the application is also useful to help a deeper analysis in the future.

The vision conceived is described in Figures 3.1 and 3.2.

Initially, the system is expected to be composed as follows:

- **Internet Application** - The CORQ application will be available on the Internet from a browser where Collaborators, Project Managers, and Directors will be able to consult and explore information on the metrics generated in the project/commit, per collaborator.

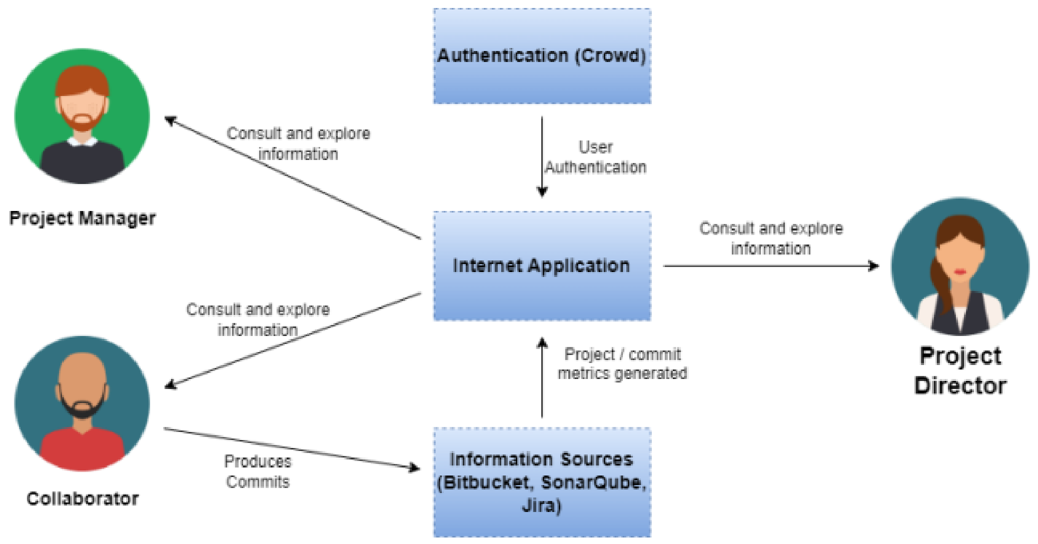

Authentication (Crowd)
User Authentication
Consult and explore information
Project Manager
Internet Application
Consult and explore information
Project Director
Consult and explore information
Project / commit metrics generated
Collaborator
Produces Commits
Information Sources (Bitbucket, SonarQube, Jira)

Figure 3.1: Initial global vision

- **Information Sources (Bitbucket, SonarQube, Jira)** - Sources of code analysis information in this case of a project/commit/time spent on problems to solve, which will be sent to the application, to be used in it.
- **Authentication (Crowd)** - Utility to validate users in application from Crowd. Crowd will provide that Opensoft Collaborators do not have to log in every time they access the application as they are already logged in on other Atlassian platforms (Jira, Confluence).

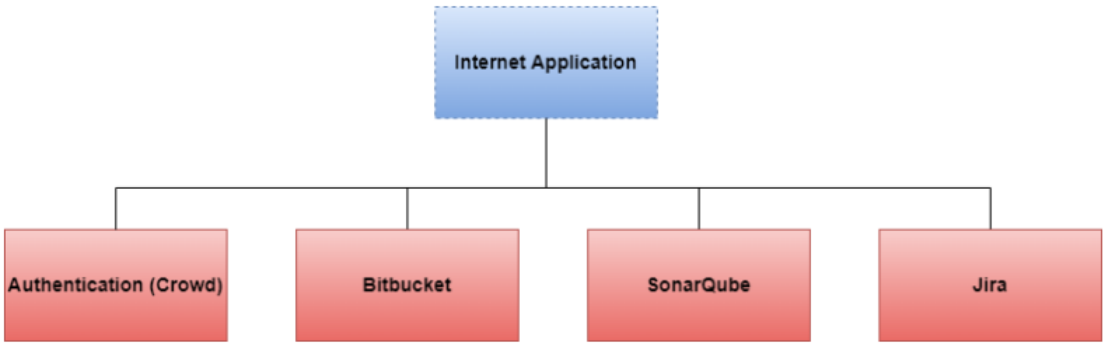


Figure 3.2: Collaboration diagram

The internet application interacts with the following systems:

- **Bitbucket** - Hosts the Git projects developed by Opensoft. It will be used in conjunction with SonarQube from its APIs to obtain information about code analysis produced by Opensoft employees.

- **SonarQube** - Static code analysis tool that provides various metrics about code quality. It will be used together with Bitbucket from its APIs to obtain information about code analysis produced by Opensoft employees.
- **Jira** - Task and time monitoring system associated with a project. It will be used from its API to get information about the time spent working by each employee.

3.4 Requirements

The necessary conditions identified have been consolidated and recognized in the set of requirements described below, grouped by category.

The previously identified needs were consolidated and recognized in the set of requirements identified below, grouped by component in the functional requirements as we can see in Table 3.3.

ID ND	ID RQ	Description	Priority
ND01	RQ01	Obtain and record information on productivity indicators by employee.	1
ND02	RQ02	Obtain and record information on quality indicators by employee.	
ND03	RQ03	Consultation and exploration of information on the company's quality and productivity indicators by employee.	2
	RQ04	Consultation and exploration of information on quality and productivity indicators of the company by indicator.	
	RQ05	Consultation and exploration of information on quality and productivity indicators of the company in general.	

Table 3.3: Functional requirements

The priority for the development of each requirement was determined by taking into account its benefit to the end user, its criticality (fundamental, important, or just useful), effort, and risk.

After the functional requirements, we have the following categories:

- **Performance Requirements** (Table 3.4): Refers to the system's performance (response times and capacity).
- **Reliability Requirements** (Table 3.5): How reliable the system is in terms of availability.
- **Maintenance Requirements** (Table 3.6): Refers to scalability, modularity and extensibility of the system.
- **System Requirements** (Table 3.7): Refers to what kind of system CORQ needs to run. In this case browser, and its application server.

- **Security Requirements** (Table 3.8): Security of the system, refers to authentication, communication, protection against attacks, monitoring, and audit.
- **Usability Requirements** (Table 3.9): Requirements refered mainly to frontend and usability.

ID ND	**ID RQ**	**Requirement**	**Description**
ND08	RP01	Response times	The processing time for each user request must be on average less than two seconds and the maximum time cannot exceed one minute for longer options.
	RP02	Capacity to use	The solution must be sized to support 100 concurrent users without performance degradation.

Table 3.4: Performance requirements

ID ND	**ID RQ**	**Requirement**	**Description**
ND07	RF01	System availability	The system was designed to allow permanent use, at any time of the day and day of the week.
	RF02	Availability of information	Information will be updated every 24 hours.

Table 3.5: Reliability requirements

ID ND	**ID RQ**	**Requirement**	**Description**
ND04	RM01	Scalability	Making the platform can be installed and used in something as simple as a laptop and even the most complex data center so that it can be fully distributed.
	RM02	Modularity	Possibility of reusing components for other functionalities, without the need to reformulate the code structure.
	RM03	Extensibility	Possibility of extending new functionalities and/or new indicators to the system, without the need to reformulate the code structure.

Table 3.6: Maintenance requirements

ID ND	**ID RQ**	**Requirement**	**Description**
ND09	RY01	Browser	The browsers the solution will support are Chrome 90+ and Firefox 90+.
	RY02	Application Server	Apache Tomcat Server version 10 that allows the execution of code developed in Java 11.

Table 3.7: System requirements

ID ND	ID RQ	Requirement	Description
ND05	RS01	Authentication	Users will have to previously authenticate to Crowd (or another application for example Jira, Confluence, etc.) to access the application's features.
	RS02	Secure communication	All communications between the Application and browsers or the external systems involved must be encrypted using secure protocols and allow for the evolution of the protocol used.
	RS03	Protection against web attacks	The solution must be protected from attacks including XSS, CSRF and SQL Injection.
	RS04	Monitoring	Keep the activity record (log) of all operations, who did it, what main and necessary data for screening were used.
	RS05	Audit	The auditability of all operations carried out must be guaranteed. In particular, all operations associated with the user who performed them, as well as the respective date/time, must be persistently recorded.

Table 3.8: Security requirements

ID ND	ID RQ	Requirement	Description
ND05	RU01	Application simplicity	The application must allow for simple, logical and intuitive navigation.
	RU02	Attractiveness of the solution	The solution must have an attractive and consistent graphic design.
	RU03	Accessibility	Respect accessibility rules in accordance with the accessibility guidelines established by the W3C.
	RU04	Responsive	The solution must ensure correct presentation on different screen sizes (smartphones).
	RU05	No download dependencies	The solution should only use the standard mechanisms included in the browsers, and it is not necessary to previously install any other software to use the solution's features.
	RU06	Information export	All features that produce tabular results (listings) should allow exporting them in Excel format (CSV and XLSX).

Table 3.9: Usability requirements

Chapter 4

Design

After analyzing what the application needs and requires, a well-structured design before the implementation is important. There are important decisions and perspectives on the application that can save time and make the implementation much clearer.

Although an initial design can look a certain way after the implementation starts the application might, and most probably will need some changes design-wise, and that is the case with CORQ.

Our view on CORQ had based on a Batch and Web application, and while developing and implementing CORQ the need to change the design and implementation of these components appeared.

CORQ went from having those two base components to having an *Elasticsearch Logstash Kibana* (ELK) stack and Filebeats, saving time in frontend development, and providing a great interface and visualizer of the data generated by the Batch component.

CORQ has a server machine where it hosts the CORQ database and ELK stack. Whenever we want to generate information about metrics of projects that are hosted in Bitbucket we can use CORQ at the server machine. If a collaborator wants to analyze and generate metrics of a project that is not hosted in Bitbucket, they can use CORQ on their computer, if they have the git repository corresponding to the project on their computer. The connection between the collaborator's version and the CORQ server's database and elastic search stack is done using SSH tunnels.

The different views and changes to the design of the application will be discussed below.

4.1 Logical Perspective

The logical architecture primarily supports the functional requirements, decomposing the system into solution structures that respond to the requirements identified by the customer.

These structures are illustrated in the following figure with the representation of the solution components and the external systems with which they interact as shown in Figure 4.1.

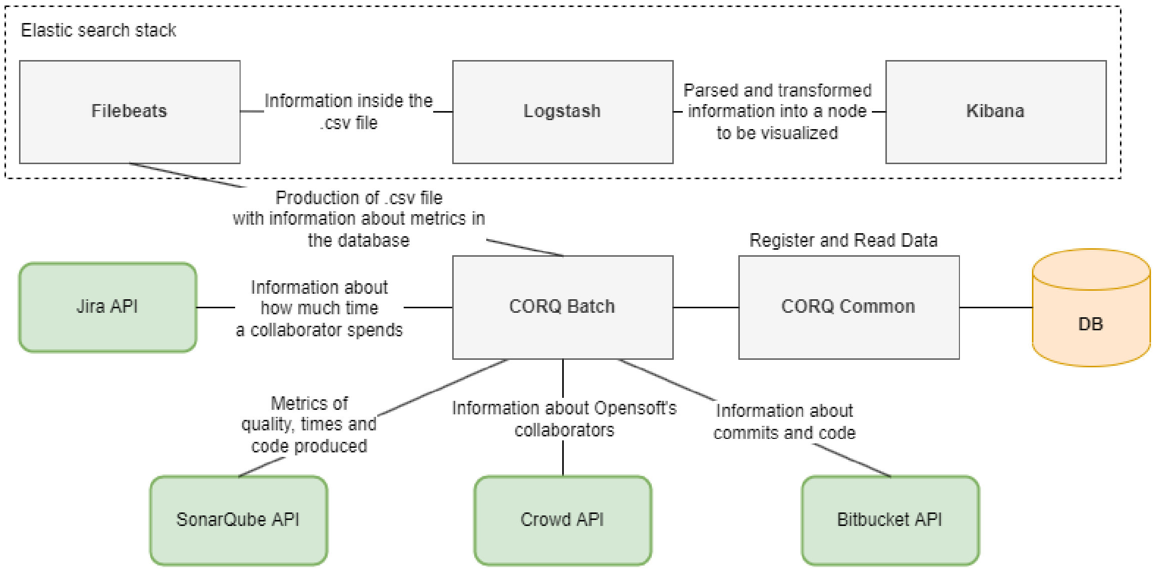

Elastic search stack
Filebeats
Information inside the .csv file
Logstash
Parsed and transformed information into a node to be visualized
Kibana
Production of .csv file with information about metrics in the database
Register and Read Data
Jira API
Information about how much time a collaborator spends
CORQ Batch
CORQ Common
DB
Metrics of quality, times and code produced
Information about Opensoft's collaborators
Information about commits and code
SonarQube API
Crowd API
Bitbucket API

Figure 4.1: Logical perspective

The solution consists of the following components, described in the components annex:

- **Kibana**, an elastic search component that is responsible for the visualization of the information produced by CORQ Batch in graphs.
- **Logstash**, an elastic search component that is responsible for the collection, transforming, and parsing of the information received from CORQ Batch, which is received from a collaborator's machine using Filebeats to publish it to the server.
- **Filebeats**, an elastic search component that is responsible for sending the information produced by CORQ Batch in a collaborator's computer to the elastic search stack at the server.
- **CORQ Batch**, a batch application that is logically designed for database population operations from external APIs used by Opensoft, schedules jobs needed to generate daily analysis, process a high volume of data, and produce artifacts such as .csv files.
- **CORQ Common**, the CORQ spring application that is logically designed for operations to read and record data in the database. Component common to all current and future CORQ components.

These applications communicate, at different times, with other external systems, as described in the interface annex:

- **SonarQube API** to obtain information on quality metrics about the analyzed code, source code, and possibly (hypothetical) time it will take for a contributor to resolve an issue.
- **Bitbucket API** to get information about commits/differences between commits of a project in question and source code if necessary.
- **Jira API** to get information about the time a contributor spends on certain project issues in Jira.
- **Crowd API** to obtain information about Opensoft collaborators.
- **Database (DB)** for manipulating the data structures that feed the system.

4.2 Implementation Perspective

The implementation perspective focuses on the organization and hierarchy of software layers of the solution as shown in Figure 4.2.

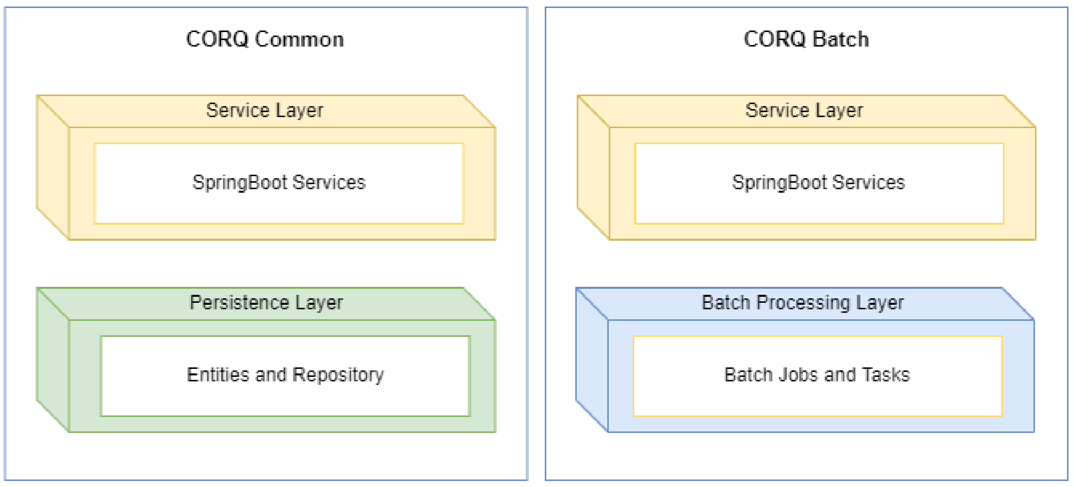


Figure 4.2: Implementation perspective

The solution layers are summarized as:

- **Services Layer**: contains the business logic structured in high-level methods oriented to User Stories scenarios. Services made available in this layer provide public APIs and communicate with external resources.
- **Persistence Layer**: the layer of the web application that represents the Database entities for communication with it. This layer acts as a medium that provides interfaces to control various data objects and abstracts technical details about data structures and their storage. The data access layer knows how and where persistent entities are stored.

- **Batch Processing Layer**: the layer of the batch processing that represents tasks and jobs implemented. This layer is responsible for processing high amounts of data that is inside the database and producing artifacts such as .csv files.

The presentation of the application is done by the ELK stack, which does not need implementation, only configuration.

4.3 Physical perspective

The physical perspective illustrates the solution's execution environment, including the network and software components that make up the system, as well as the communication between them.

This perspective allows evaluation of non-functional performance, scalability or reliability requirements, and the network topology where the solution will be installed.

4.3.1 Deployment

The deployment of versions will be carried out with a ZIP file containing the CORQ Batch module.

Figure 4.3 illustrates the topology of the network on which the solution is installed. We can visualize every layer of implementation from a physical perspective, and how every physical artifact interacts with each other.

From top to bottom, we can see a user accessing https://corq.opensoft.pt on the internet with the credentials needed. Kibana works as a proxy to the rest of the ELK stack, forwarding any request from https://corq.opensoft.pt. The ELK stack contains information that was generated by CORQ Batch.

In terms of the backend and Data layer access, CORQ Batch generates metrics by communicating with the SonarQube, Bitbucket, and Jira APIs with REST calls using Retrofit2 and OkHttp3.

It then communicates with the MariaDB Database on CORQ's Schema using JDBC to write and read data.

The hosting of CORQ was configured and implemented by my supervisor at Opensoft.

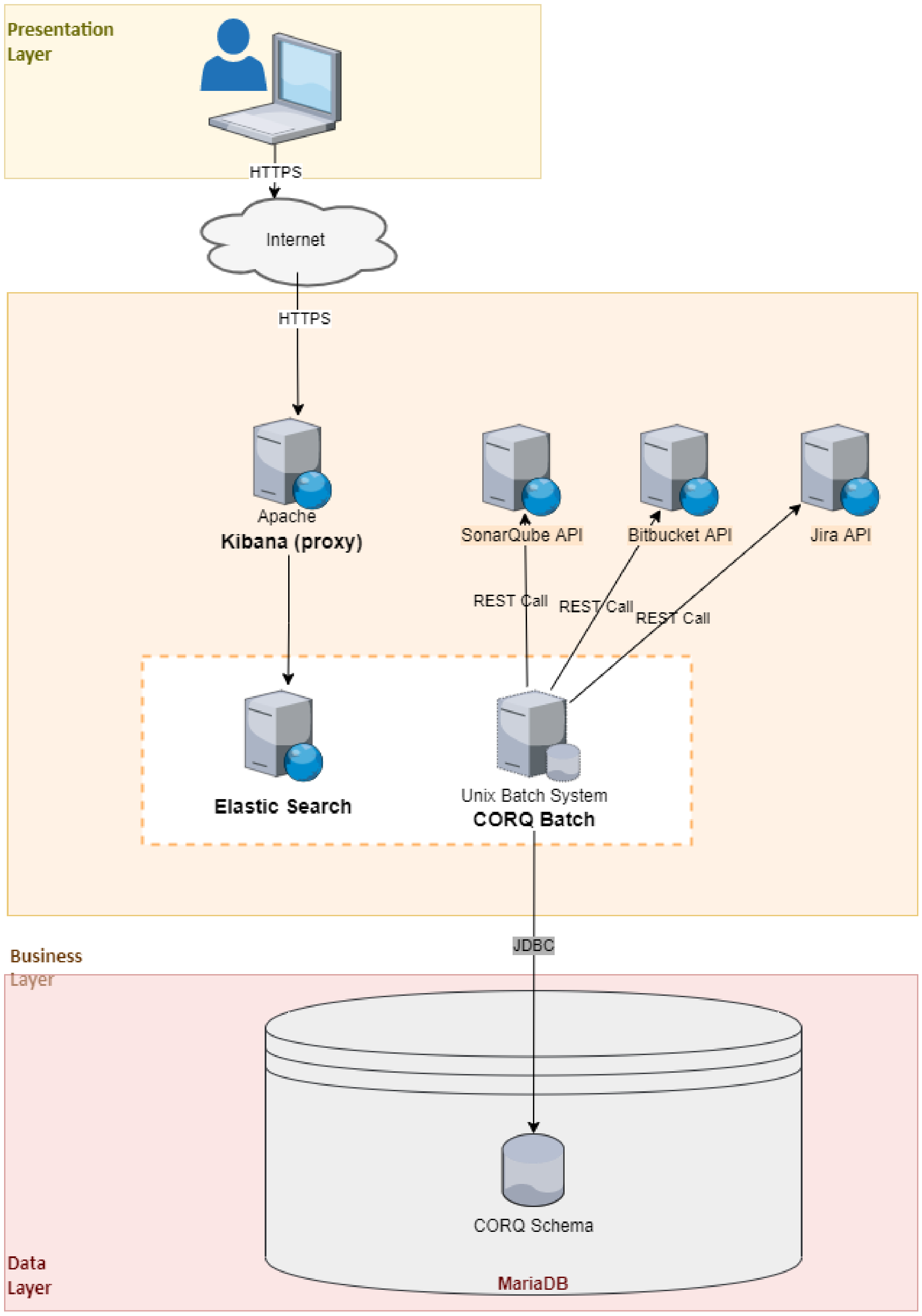


Figure 4.3: Physical deployment

4.4 Components

Components are low-level entities of the solution and are integrated to produce the product.

These entities implement one or more activities, can provide or require interfaces with other components, and can be services, systems, or even sub-components.

CORQ Common is a Spring sub-component used by CORQ Batch and any other component that might be included in the future. It contains classes and services that are used commonly throughout the application, having a service and a persistence layer. This sub-component uses frameworks such as Spring boot, Apache Maven, and JPA. It is implemented in Java 11 and SQL. For logging, it uses SLF4J and Logback and follows an MVC architecture. For middleware it uses Apache HTTP Server and Apache Tomcat as it is the default for Spring applications.

CORQ Batch is a Spring Batch component that runs in the CORQ application, available on a Unix system. The purpose of this component is to populate the database with new data and read and update existing data. This component uses frameworks such as Spring Batch, Apache Maven, Retrofit2, OkHttpClient3, and Lombok. It is implemented in Java 11. For logging, it uses SLF4J and Logback and follows an MVC architecture. For middleware it uses Apache HTTP Server and Apache Tomcat as it is the default for Spring applications.

4.5 Communication Interfaces

The communication interfaces used were **REST, HTTP, and JDBC**. These technologies were described in the second chapter.

The only interface that needed some kind of decision-making on what we were going to use was to decide between REST and SOAP since HTTP is the standard protocol typically used by browsers to communicate with application servers and obtain resources such as HTML documents, and JDBC is the standard communication interface to communicate with databases in Java.

After analyzing the pros and cons of REST and SOAP, the decision to use REST is based on these factors as cited in [9] and [10]:

- **SOAP is a language, platform, and transport independent option** (does not enforce HTTP); standard; built considering errors. Higher implementation costs, greater benefits in terms of being more standard and documented, and less risk of technical errors.
- **REST is easier to use and flexible; efficient; fast**. It becomes more efficient but could bring more risks.
- **REST is more commonly used these days**, making **SOAP harder to use** to consume the APIs that are required in the case of **CORQ**.

After deciding to use REST, some frameworks had to be used to access and manipulate APIs for the information we need.

The decision towards the framework used to make REST API requests to outer interfaces has been to use **Retrofit2 with an OkHttp3 Client.** Retrofit2 is a framework built on top of OkHttp3, both being developed by the same company. These two prove to be more efficient and easy to use and implement in any situation.

In comparison with other web client implementations such as Spring's WebClient or RestTemplate, using **Retrofit2 with OkHttp3 client shows to be beneficial in many ways**.

Retrofit2 is a web client implementation built on top of OkHttp3, it is an implementation to simplify code, and how easy it is to use [11]. So in terms of benefits, Retrofit2 has the same benefits as OkHttp3 has in comparison to other clients usable in Java, making it easy to learn and develop.

OkHttp3 supports Sync and Async calls, HTTP/2, Forms, Multipart/file upload, Cookies, Authentication, GZip Deflate transparent content compression, Response Caching, and Web Sockets. OkHttp3 shows the most benefits out of the clients available at the moment [12] [13], so it was the framework decided to be used.

4.6 Data Model

Although simple, the CORQ data model has been through some changes in the development of this application. The entities to consider in our project are the Collaborators, Projects, and the Metric measurements taken from them.

The initial idea was that we had Opensoft's users, projects, and their relationship table UserProject, which contains information on a user and a project, so this way we could map all users in a project and all projects a user is a part of. This is shown in Figure 4.4.

The Indicator object is the most complex entity we have in this model. Indicators will be generated daily.

We would have two attributes for each metric we wanted to measure.

Indicators would have the raw value and a rank from F to A. Type is ”User”, ”User-Project” or ”Project”.

Daily means the indicator generated daily for a collaborator, project, or user in a project. Overall means indicator for the user, project, or user in a project for the whole time being.

This also updates daily after we generate the daily indicator for the same user, project, or user in a project. Then we have the date of analysis and daily status. Daily status is an attribute for ”overall” indicators, which states how the overall indicator changed depending on the daily indicator.

After some thought to make this data model more extensible, easy to understand, and

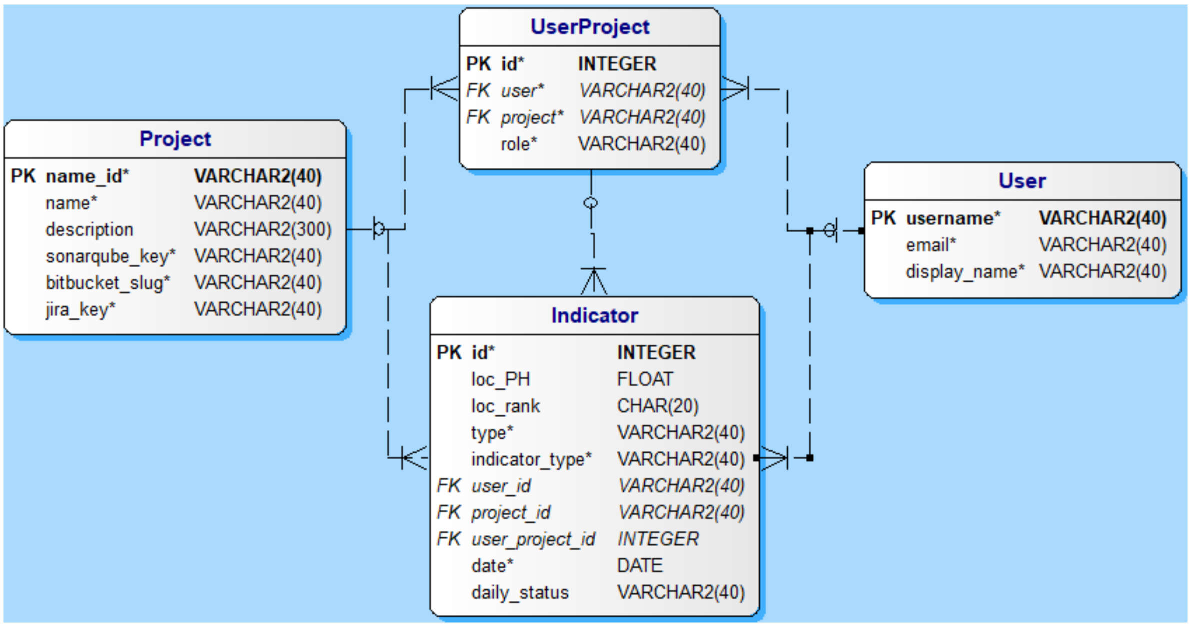


Figure 4.4: First data model

use, this was the new and final data model as shown in Figure 4.5.

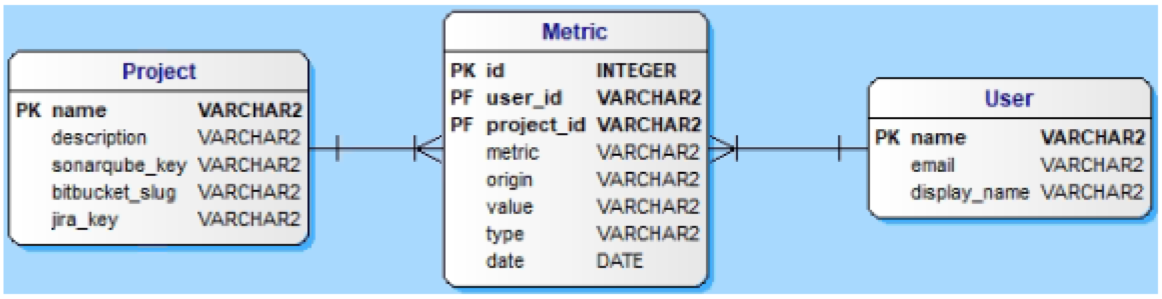


Figure 4.5: Final data model

In this data model, the relationship between users and projects is included in the new entity called ”Metric” instead of ”Indicator”. The ”Project” and ”User” tables did not go through many changes.

The main change here was the new entity to show the measurement metrics. A metric now refers to a specific ”Metric” - as shown in one of the attributes.

The metric attribute refers to an enumeration that is referring to a specific metric defined in the system. It is shown as a VARCHAR but is stored in Java as an enumerate that can be extended to any new metric we want to add.

It can be for example ”LINES_OF_CODE”, ”BUGS”, ”MAINTAINABILITY” or ”RELIABILITY”.

The origin of the metric refers to the API where we get that information, that being SonarQube, Bitbucket, or Jira. It is shown as a VARCHAR but is stored in Java as an enumerate that can be extended to any origin we want to add.

The value depends on the origin and metric, which can be a string, integer, or something else. If the origin is ”Bitbucket” for example, then we would convert the value from VARCHAR to integer, since the value of a metric coming from Bitbucket will be lines of code for now.

The type attribute refers to the type of value, as explained before. The value of the metric is always stored as a VARCHAR but is converted to the type in this attribute. It is shown as a VARCHAR but is stored in Java as an enumerate that can be extended to any value type we want to add.

The database is relational as of now, but it might be changed in the future if the change makes sense. For now, this is the safest option, but the investigation in this regard has already been done, and it might be an option to change to a *NoSQL* database in the future.

4.7 User stories

User stories are fundamental when developing an application. They give us a good and easy way to understand the description of a real problem, of a real stakeholder. It would be too extensive to describe all user stories made for this application, so the description of what they represent and an example of a user story documented will be given.

- **Title**: CORQ User Story USXX - As a <stakeholder> I want to <description of action>.

  Example: CORQ User Story US01 - As System, I intend to generate productivity indicators for an employee.

- **Description**: Scenario description.

  Example: It is intended that the system, from accessing external APIs such as SonarQube, Jira, and Bitbucket, obtain the necessary information about the code produced by collaborators, and from this information, it can generate information on productivity indicators of that collaborator.

- **Business Rules**: Rules made in terms of business.

  Example as shown in Table 4.1:

RN ID	**Rule**
RN01	Information on these indicators is updated every 24 hours.
RN02	The external interfaces must be available so that we can obtain information about the collaborators in them.

Table 4.1: Business Rules Example

- **Inputs**: Inputs needed to be given by the stakeholder.

  Example: The inputs given to the System in this user story will be given from the external interfaces used by Opensoft, such as the time the collaborator spent to make a certain code and its quality.

- **Outputs**: Outputs needed to be given by the stakeholder.

  Example: The output of this user story will be the population of the System's database with the productivity indicators on employees, calculated therein.

- **Support Diagram (Optional)**: Diagram made to ease the understandability of the user story.

- **Interfaces**: Information about what systems this system must interface, constraints, or protocols used.

  Example as shown in Table 4.2:

Layer	System	Service	Description
Backend	Jira	/Jira/	REST interface used by the backend to obtain information to generate indicators
	SonarQube	/sonarqube/	
	Bitbucket	/bitbucket/	

Table 4.2: Outside Interfaces Example

- **Acceptance Criteria**: Criteria that need to be met for system acceptance.

  Example as shown in Table 4.3:

CA ID	Description
CA1	**If** the Employee worked in a certain period of time **then** there are quality indicators about this employee in the period of time in question to calculate the new indicators.
CA2	**If** the Employee **did not** work in a certain period of time **then** there are **no** quality indicators to be calculated in that period of time.

Table 4.3: Acceptance Criteria Example

4.8 Project structure

The project structure follows a Spring and MVC convention.
Packages of corq-common:

- **src/main/java/pt/opensoft/corq/common/common/:** package containing classes that are commonly used in this module, this includes a configuration file to set beans used by retrofit2 in every outside interface access retrofit2 instance.

- **src/main/java/pt/opensoft/corq/common/model/:** package for util data models.
- **src/main/java/pt/opensoft/corq/common/persistence/entity/:** corq database entities implementation.
- **src/main/java/pt/opensoft/corq/common/persistence/repository/:** corq database repository implementation, this includes queries used to access or manipulate specific data of the database.
- **src/main/java/pt/opensoft/corq/common/service/:** package with corq common service implementation, this service contains all logic of methods used to manipulate classes in this module, including database access and manipulation.
- **src/test/java/integration/entity/:** tests on corq entities.
- **src/test/java/integration/repository/:** tests on corq entities.
- **src/test/java/integration/service/:** tests on the corq common service.
- **src/test/resources/:** configuration files.
- **target/:** folder where any construct artifacts and output from execution go.

Packages of corq-batch:

- **bin/:** contains binary files, which are executable files that the user or server executes to accomplish a certain purpose. The binary files produced will be described below.
- **filebeats/:** - folder containing all files needed by Filebeats, these files are used by some binary files to publish produced artifacts to the server by using an SSH tunnel.
- **logs/:** log files of CORQ, used for debug purposes.
- **repositories/:** - empty folder to be used in case the user decides to clone projects from the Bitbucket into the local machine, user, or server and decides to use this folder.
- **src/assembly/:** files used to describe how to assemble the artifacts that are produced by CORQ, used by a plugin in the pom.xml file.
- **src/main/java/pt/opensoft/corq/batch/config/:** config files used when assembling the project into a zip file.
- **src/main/java/pt/opensoft/corq/batch/interfaces/:** package containing services and retrofit2 instances of each outside interface.

- **src/main/java/pt/opensoft/corq/batch/jobs/:** package containing all corq batch jobs implemented, each job is in a separate package containing its config class and tasklet classes.
- **src/main/java/pt/opensoft/corq/batch/service/:** package with corq batch service implementation, this service contains all logic of methods needed in the corq batch jobs.
- **src/main/java/pt/opensoft/corq/batch/util/:** package containing util classes used in corq batch jobs including email utils, git utils, and key helpers for data structures.
- **src/main/profiles/:** configuration profiles used by maven and spring.
- **src/main/resources/:** configuration files and HTML file used in e-mail sending.
- **src/test/java/integration/:** integration tests for corq batch jobs.
- **src/test/java/service/:** tests on the corq batch services.
- **src/test/resources/:** test configuration files.
- **target/:** folder where any construct artifacts and output from execution go.

Chapter 5

Implementation

CORQ has been developed following Opensoft's technologies and approaches to their applications in order to have a solid, efficient, and optimized Java application. The implementation for CORQ happened mostly as planned. The implementation and changes made to it will be described in this chapter.

CORQ was implemented using the base tools used at Opensoft as described in the introduction, with Maven as a build automation tool, Java as the base programming language, Spring as the main framework to develop in Java, and MySQL (MariaDB) database. The addition of the ELK stack for the front end was a decision made after the initial design was done, and at a later phase of the implementation.

Spring has sub-frameworks that follow the MVC pattern, which makes all the components that were planned to be implemented work well together.

Another important framework used was JGit. It is a framework that helps us access and manipulated a git repository as long as we have that repository in the local machine. It is the main piece used in the Spring batch component to process every information we need in Opensoft's projects that are inBitbucket and SVN once they exist in the local machine.

For logging SL4J was the framework used.

In terms of API access and manipulation, the tools used were Retrofit2 and OkHttp3 as explained in the chapters prior.

In the persistence layer, the main frameworks used were Java Persistence API (JPA) and the MariaDB Java Client. Spring connects with JDBC using the MariaDB Java Client and uses JPA to represent the entities and their connections to each other in Java, and these Java entities are mapped into the database.

For quality assurance, some Maven frameworks were also used, and SonarQube analysis on a weekly basis.

The deployment of the project was done with Jenkins, and the artifacts produced were used in a server machine and collaborator's machines if they decide to add to CORQ a project that is not existent in Opensoft's Bitbucket platform. The configuration of CORQ in Jenkins was made by my supervisor.

For now, the implementation of the project only produces three metrics, those being lines added, lines removed and lines changed. These metrics are generated by using the JGit framework and by accessing Bitbucket's API. Other metrics that were planned to be implemented were not because of time resources available. Although more metrics were not implemented, the implementation of the project had a big focus on extensibility so the implementation of new metrics is the easiest possible. This includes the study of Jira, Confluence, and SonarQube APIs.

5.1 First month

Although the first month at the company was planned to be the analysis and design of the project, it was also to understand and learn the methodology of the company. In the first week at the company, I was part of an introductory workshop for the whole week. I participated in Java, SQL, Spring, and Tools modules. It gave me an idea of the big picture of how projects are implemented at the company and the mindset behind every line of code.

After the first week, I was introduced to my supervisor and the software architect I would work alongside and started working on the analysis of tools and ideas for CORQ. From then on, I had reunions twice a week with them to have an idea of what should be done and how to organize my work while implementing CORQ.

At the same time as I was analyzing tools and having ideas for CORQ, I started exploring tools for API consumption such as Retrofit2 and exploring the Bitbucket and SonarQube APIs, since these two were considered the most interesting to have the first metric indicators.

5.2 Project base configuration

Initial configurations for the project started with the Bitbucket platform. A version controlling system is fundamental for a good development process. The project was also set up in Confluence and Jira to document anything needed and to keep track of the time used in specific tasks of the project.

5.2.1 Maven configuration

The maven configuration is based on an XML file called POM (Project Object Model). It is an XML file that contains information about the project and configuration details used by Maven to build the project. The configuration of Maven for CORQ is the following:

The project itself has a pom.xml file, and each of the modules has its pom.xml file as well.

For the main project pom.xml file, the main configuration lines needed are:

- <modules> - indicate the modules that this project includes.
- <groupId> - group id, which in our case is pt.opensoft.
- <artifactId> - project id (corq).
- <version> - current version of the project. For corq, we follow semantic versioning.
- <name> - name of the project.
- <description> - brief description of the project.
- <packaging> - what primary artifact is done with this project - we use ”pom” for the main project.
- <scm> - define what CVS system is used and its URL (Bitbucket and the link to the project in Opensoft’s Bitbucket platform).
- <issueManagement> - define what issue management system is used and its URL (Jira and the link to the project in Opensoft’s Jira platform).
- <properties> - maven properties used, in this tag we define the version of maven used to compile.
- <reporting> - define plugins that are executed during the site generation.
- <build> - define plugins that are executed during the build.
- <dependencyManagement> - define dependencies used in all the modules defined earlier.
- <repositories> - define connections to repositories from which we will use dependencies or other poms.
- <pluginRepositories> - define connections to repositories from which we will use plugins.
- <distributionManagement> - specifies where and how this project will get to a remote repository when it is deployed.

Example:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
     https://maven.apache.org/xsd/maven-4.0.0.xsd">

  <modelVersion>4.0.0</modelVersion>

  <modules>
    <module>corq-batch</module>
    <module>corq-common</module>
  </modules>

  <groupId>pt.opensoft</groupId>
  <artifactId>corq</artifactId>
  <version>0.1.0</version>

  <name>corq</name>
  <description>CORQ - Code Review for Quality
     Measurement</description>
  <packaging>pom</packaging>

  <scm>
    ..
  </scm>

  <issueManagement>
    ..
  </issueManagement>

  <properties>
    <maven.compiler.source>11</maven.compiler.source>
    <maven.compiler.target>11</maven.compiler.target>
  </properties>

  <reporting>
    <plugins>
      ..
    </plugins>
  </reporting>

  <build>
    <plugins>
      ..
    </plugins>
  </build>
  <dependencyManagement>
    <dependencies>
      ..
    </dependencies>
  </dependencyManagement>

  <repositories>
    <repository>
      ..
    </repository>
  </repositories>

  <pluginRepositories>
    <pluginRepository>
      ..
    </pluginRepository>
  </pluginRepositories>

  <distributionManagement>
    <repository>
      ..
    </repository>

    <snapshotRepository>
      ..
    </snapshotRepository>
  </distributionManagement>

</project>
```

For the modules that are children of the main project module, which are corq-common and corq-batch, the following tags are different or new:

- <parent> - indicate the module's parent by indicating what is the group id, artifact id, and version of the parent.

- <groupId> - we do not define a group id since it is the same as the parent's, we only artifact id which is the name of the module.

- <profiles> - define different profiles used for the project such as local, development, or production.

- <dependencies> - indicate what dependencies from the parent module, this module will use.

5.2.2 Spring configuration

To configure spring in our maven application we need to add spring's dependencies in our pom.xml file. The versions indicated were the ones used at the end of the development, the versions were being updated to the LTS (Long Term Support) version at the time. The main dependencies are:

- **Spring boot** version 2.6.7 - Spring boot helps create stand-alone Spring applications, has embedded Tomcat, provides opinionated 'starter' dependencies to simplify your build configuration, automatically configures Spring and 3rd party libraries whenever possible, provides production-ready features such as metrics, health checks, and externalized configuration and requires no XML configuration.

- **Spring batch** version 4.3.6 - A lightweight, comprehensive batch framework designed to enable the development of robust batch applications vital for the daily operations of enterprise systems. Spring Batch provides reusable functions that are essential in processing large volumes of records, including logging/tracing, transaction management, job processing statistics, job restart, skip, and resource management. It also provides more advanced technical services and features that will enable extremely high-volume and high-performance batch jobs through optimization and partitioning techniques. Used to process high volumes of git branches and commits.

- **Spring data JPA** version 2.6.7 - Spring Data JPA, part of the larger Spring Data family, makes it easy to easily implement JPA-based repositories. This module deals with enhanced support for JPA-based data access layers. It makes it easier to build Spring-powered applications that use data access technologies. This framework is used to connect our Spring application to the database and manipulate its JPA entities and repositories.

- **Spring mail** version 2.6.7 - Utility library for sending an email that shields the user from the specifics of the underlying mailing system and is responsible for low-level resource handling on behalf of the client. This framework is used to send e-mails containing the metrics to the managers at Opensoft.

The only configurations that are needed for these frameworks are some options in the "application.properties" files, and annotations used in code.

The configurations in the code for the module main classes are described in comments (green text) after each annotation.

For corq-batch's main class, the configurations used were:

```
@Configuration // Allows registering extra beans in the context
   or importing additional configuration classes.
@ComponentScan(basePackages = {"pt.opensoft.corq.common",
   "pt.opensoft.corq.batch"}) // Scans components in
   corq-common module and on this one.
@EnableJpaRepositories(basePackages =
   {"pt.opensoft.corq.common"},enableDefaultTransactions =
   false) // Enables JPA repositories in corq-common, and
   disables default transactions.
@EntityScan(basePackages = {"pt.opensoft.corq.common"}) //
   Scans entities in corq-common package since it is contains
   the persistence layer.
@PropertySource(value = "classpath:corq.local.properties") //
   Used to provide properties file to Spring Environment.
public class CorqBatch {

   ..

}
```

For corq-common's main class the configurations used were:

```
@SpringBootApplication /* This annotation enables these three
   annotations:
 - @EnableAutoConfiguration: enable Spring Boots
    auto-configuration mechanism
 - @ComponentScan: enable @Component scan on the package where
    the application is located (see the best practices)
 - @Configuration: allows to register extra beans in the context
    or import additional configuration classes*/
public class CorqCommon {

   ..

}
```

To define properties that we want to use inside our Java code, Spring provides the possibility to use an ”applications.properties” file to define the properties. After that file exists, if we want to use any property in that file inside our code, we have to implement a configuration class that has getters to each of the properties.

These files can be different depending on which profile we are in. Profiles are a strategy to differentiate the environment in which we are using the application. For example, if we are using the application in a development environment, we would use a different database than if we are using the application in production. By using a different database, we make sure that the database used by the application by users in real-time is not affected by any changes done by the developer.

This way, we can have a development environment with a database that is a replica of what the database used by users in real-time to test our code in an environment as close to the reality of our application as possible.

Example of the applcation.properties file used in the development environment, showing only properties used automatically by spring:

```
server.port=8080
spring.datasource.url=jdbc:mariadb://localhost:3307/corq_local_db
spring.datasource.username=dev_user
spring.datasource.password=dev_pw
spring.datasource.driver-class-name=org.mariadb.jdbc.Driver
spring.jpa.hibernate.ddl-auto=update
spring.jpa.generate-ddl=true

spring.mail.host=smtp.gmail.com
spring.mail.port=587
spring.mail.username=corq@opensoft.pt
spring.mail.password=corq_pw
spring.mail.properties.mail.smtp.auth=true
spring.mail.properties.mail.smtp.starttls.enable=true
mail.smtp.debug=true
```

The database configuration is done with Spring and JPA making it an automatic process. JPA will create the database in MariaDB corresponding to entities, repositories, and relations between entities defined in our code with annotations. The dependency used for this purpose was **MariaDB Java Client** version 2.7.4.

Other properties were used for implementing purposes but these are configuration properties that the spring needs to function.

5.3 Outside APIs

CORQ consumes the APIs of the platforms used by Opensoft from which we need information on collaborators and projects. Each subsection below will explain why we need to consume that API and how it is done. Although four APIs were explored and set up for this application, for now only the Bitbucket API is being used to generate and process the lines of code added, changed, and removed indicators. The others were set up and explored at the start of the project and are documented and ready for whenever they will need to be implemented.

Requirements for every outside API:

- The application assumes 24/7 interface availability
- The interface assumes the application access with specific credentials for it.

An extra requirement for the database is that communication is made via JDBC.

5.3.1 Bitbucket

This tool's API is used to get some information about contributor commits as shown in Table 5.1.

Type	Path	Description
GET	/users/{selected_user}	API's selected user's information
	/repositories	API's selected user's information
	/repositories/{workspace}	API's listing of all repositories of a specific workspace can be queried on a specific page
	/repositories/{workspace}/ {repo_slug}	API's repository information
	/repositories/{workspace}/ {repo_slug}/commits	API's repository commits page, can be queried in a specific page
	/repositories/{workspace}/ {repo_slug}/commits	API's repository commits page
	/repositories/{workspace}/ {repo_slug}/commit/{commit}	API's repository specific commit
	/repositories/{workspace}/ {repo_slug}/diffstat/{spec}	Get information about the difference between two commits
	/repositories/{workspace}/ {repo_slug}/src/{commit}/ pom.xml	Get raw code of the project's "pom.xml" file
POST	/site/oauth2/access_token	Get access tokens to Bitbucket's API

Table 5.1: Bitbucket API Requests

Bitbucket's authentication is based on OAuth 2.0. This way, authentication was implemented to get an access token that expires every two hours, and a refresh token to keep getting new access tokens whenever we need them. To generate these tokens we need to create a "consumer", which is a kind of token created in Bitbucket, that gives us a key and a secret. The key and secret are used to generate both tokens. To create this consumer we need admin permissions in a specific workspace.

Base URL to get tokens: https:/bitbucket.org/

Authentication: Basic authentication with user credentials.

Base URL for API access: https://api.bitbucket.org/2.0/

Authentication: Bearer token with an access token.

POST call to get access tokens can have different Header fields to get the access token. It can be requested with the consumer key and secret or with a refresh token, by changing the field "grant_type" and giving authorization different information based on that.

5.3.2 Jira

This tool's API is used to obtain some information about the time worked by employees on certain Jira issues as shown in Table 5.2.

Type	Path	Query	Description
GET	/search	?jql=project={project_name} &maxResults={mR}	Get project issues
	/project		Get all the API's projects

Table 5.2: Jira API Requests

Base URL for API Requests: https://jira.opensoft.pt/rest/api/2/

Authentication: Basic authentication with user credentials.

Jira might need OAuth implemented in the future but Adhoc requests as we're doing right now are helpful to implement and test what information we need from the API before implementing OAuth and integrating it into the system.

5.3.3 SonarQube

We use this API to obtain information about metrics obtained from code analysis, source code, and estimated time for an employee to resolve certain issues discovered by SonarQube as shown in Table 5.3.

Type	Path	Query	Description
GET	/issues/search	?componentKeys= {project_name}	API's issue list search by component keys
	/measures/component	?metricKeys={m1,m2} &component={c}	API's component measures
	/measures/component_tree	?metricKeys={m1,m2} &component={c}	API's component tree measures
	/projects/search		Get all the API's projects
	/projects/search	?projects={project_name}	Get one project

Table 5.3: SonarQube API Requests

Base URL for API Requests: <sonarqube_link>/api/ where <sonarqube_link> is the SonarQube instance URL. That being sonar.opensoft.pt.

Authentication: Basic authentication with user credentials.

We can also use a token generated on SonarQube's web page. The token is used also in basic authentication, with an empty password.

5.3.4 Crowd

Crowd is a centralized user system developed by Atlassian. We use this API to update and populate CORQ's user database with Opensoft's collaborator's information. The requests made to Crowd's API are described in Table 5.4.

Type	Path	Query	Description
GET	/usermanagement/1/search	?entity-type=user	Get all Crowd users
	/usermanagement/1/user	?username={username}	Get a specific Crowd user

Table 5.4: Crowd API Requests

Base URL for API Requests: https://crowd.opensoft.pt/crowd/rest/

Authentication: Basic authentication with application credentials. These credentials are created through Opensoft's Crowd.

5.4 Database

For CORQ's database, we need to use centralized platforms to populate it. We need the Users and Projects that already exist on Opensoft's platforms to give information about quality and productivity indicators on these and show them in CORQ.

For Users, we use Crowd's API. Opensoft uses Crowd as a centralized database to have information about its employees. Opensoft has centralized accounts on Crowd, so they do not have to create a new account for the same employee on every platform used. This happens due to Opensoft using Atlassian platforms all around.

For Projects, the strategy is to use specific tags in the Project's ”pom.xml” file. The ”pom.xml” or Project Object Model is the fundamental unit of work in Maven. The ”pom.xml” file is an XML file that contains information about the project and configuration details used by Maven to build the project. By telling CORQ what the Project link is in Jira and other platforms, we don't need to try and guess where it is by using its name to search it. The ”pom.xml” file reading and processing was implemented to create projects with all the keys and links to other platforms, but since most projects at Opensoft do not contain this information in their ”pom.xml” file, it was not used since only the Bitbucket metrics were implemented as of the end of the project. Generic information was used on these attributes instead.

On SonarQube's example, the Project's key is automatically set to the ”artifactId: projectname” - so for CORQ it would be ”pt.opensoft:corq”. This information is also attainable in ”pom.xml”. The information is obtained by accessing every Project in Opensoft's Bitbucket Workspace via Bitbucket's API. This way there is enough information on every Project - name and links to other platforms. The links to other platforms are useful to generate the project's quality and productivity indicators, for when they are implemented.

Projects who do not exist in Bitbucket can also be populated using one of the batch processors produced with CORQ Batch, in case it is a project that Opensoft is developing outside of the company. This will be explained further in this chapter.

5.5 Batch jobs implemented

Various batch jobs were implemented each having one purpose. They are separated into different jobs to achieve the least dependency between jobs. They can be used in various orders and combinations this way. This also makes the project more extensible, so if we want to implement a new job it is not dependent on another implementation that already exists.

The batch jobs implemented follow a Tasklet approach, this meaning that each job can do many independent tasks, and each task is implemented in a Tasklet. Each Tasklet executes the code inside the execute() function in that Tasklet. This execute function returns a RepeatStatus enumerate which can be CONTINUE or FINISHED. If it is FINISHED then the code will run the next Tasklet, or finish if there is not another Tasklet after this one, if it is CONTINUE then the execute() function of this Tasklet will execute again.

Each batch job is implemented by having a <JobName>Config.java class, which configures the factories needed, constant names needed, and the tasklets configuration.

5.5.1 TrxStats

TrxStats is a Java class implemented by Opensoft that was used as a helper for logging purposes on every batch job in CORQ. TrxStats eases the storage and structure of data as a batch job runs. An example would be the cloning repositories job. When we run this job, if TrxStats is implemented, we can integrate it into the job so that when the execution of this job is done, we can visualize data on how the execution happened, and statistics of the execution. In the example's case, we have information on:

- Total Git repositories in Bitbucket analyzed.
- Git repositories in Bitbucket cloned that did not exist locally.
- Git repositories in Bitbucket that already existed locally cloned.
- If an error happens, a brief description of the error and the stack trace are shown.

This structure is used in every job, with the total and error shown existing in every TrxStats object structure, and the other two stats being different for every different case in every job.

5.5.2 Clone Repositories Job

The main purpose of this job is to clone the repositories existing in Opensoft's Bitbucket platform into the machine that is running this job, in a folder that is specified in the properties file.

This job executes one tasklet.

This job requires that the database must already be populated with the projects we want to attempt to clone.

For each Project that exists in the database, the tasklet will get its BitbucketSlug attribute.

If a folder with that BitbucketSlug exists in the destination folder, then we will try to open that repository and fetch and pull that repository to the most recent version. If opening the repository does not work, then it will be cloned into that folder.

Else, if the folder does not exist, we will clone that Project into the machine from scratch.

5.5.3 Generate Metrics Job

The main purpose of this job is to generate metrics about the projects and users existing in the database, only Bitbucket metrics for now, by analyzing git trees of repositories that exist in the machine.

This job executes three tasklets. Since git trees can be a hassle to process, due to their complexity and size, the second tasklet uses eight threads to process the data. Thread-safe data structures were used to synchronize the threads and ensure the processing of each thread does not affect the others, and that every data that is processed is not processed twice or no times at all. Tests on how many threads should be used for maximum performance were also done.

This project requires that all projects that we want to generate metrics about exist in the machine.

GetCommitsTasklet

This tasklet's task is to analyze all projects that exist in the repositories' local machine folder and send all the commits that still need to be processed to the next tasklet.

For each folder in the projects directory, this tasklet uses the Jgit framework to process every branch's commit. The branches are analyzed one by one, in alphabetic order. For each commit, it is considered a non-processed commit a commit that:

- Still has not been processed in this execution.
- Is more recent than the most recent commit processed for this project and this branch (saved in the database).

- Is not a merge commit.

If a commit is not the most processed commit then it is saved in a synchronized thread-safe queue, which saves the commits to be processed in the next tasklet.

The most recent commit processed for each branch is also saved in a synchronized map that will be used in the last tasklet to save these processed commits into the database, for the next execution's purposes.

This way of processing each commit makes it so commits that exist in more than one branch are not processed twice, and commits that were already processed in past executions won't be processed again.

When every commit has been analyzed and sent to the processing queue if needed, the tasklet is over, and the job moves to the next tasklet.

GenerateMetricsTaskletTasklet

This tasklet's task is to analyze all commits in the thread-safe synchronized queue created by the tasklet that ran before this one. In that queue all the commits that need to be processed exist. These commits are parallel processed by eight threads, and each one processes one commit that is retrieved from the queue at a time.

With the use of the Jgit framework, we gather information about the commit such as user, project, and time of the commit.

The framework makes the difference between the commit and its parent, and CORQ associates this difference's values to a metric assigned to this user, in this project, on the day the commit was done. In each commit, we access its ”Edit” list, this means, the framework has a list of edits or differences from the commit to its parent.

The metrics generated by this task are lines added, lines removed and lines changed. To obtain an ”Edit”'s values:

1. Obtain the total value of lines added and removed in a certain ”Edit” between the commit and its parent using indexes on the file on each commit, being ”A” the commit's and ”B” the parent's.

   ```
   linesNewCommit = edit.getEndB() - edit.getBeginB();
   linesParent = edit.getEndA() - edit.getBeginA();
   ```

   Lines new commit are the lines from end to start of the ”Edit” region in the new commit. Lines parent are the lines from end to start of the ”Edit” region in the parent commit.

2. To calculate the lines added, removed, and changed we have to compare the difference between the two values calculated. If there are more lines in the new commit than the parent commit, then lines were added. If there are more lines in the parent commit than in the new commit, then lines were removed. In code:

```
linesAdded = max(linesNewCommit - linesParent, 0);
linesRemoved = max(linesParent - linesNewCommit, 0);
```

If the value of lines added is negative, then it means lines were removed, thus meaning the value goes to 0, and that is why the max() function is used. The value is always the biggest between 0 and the value calculated.

3. For lines changed, it is calculated by making the difference between the lines the new commit has added in this edit, and the lines added by the commit. In code:

```
linesChanged = linesNewCommit - linesAdded;
```

For example:

1. In this edit, in the parent, commit the file has 256 lines, the edit was between lines 45 and 55, and in the new commit the file has 266 lines, and the edit was between lines 45 and 50.
2.
```
linesNewCommit = 55 - 45 = 10
linesParent = 50 - 45 = 5
linesAdded = max(10 - 5, 0) = max(5, 0) = 5
linesRemoved = max(5 - 10, 0) = max(-5, 0) = 0
linesChanged = 10 - 5 = 5
```

3. In this edit, out of the ten lines in the new commit, 5 were changed, and 5 were added.

The values calculated are merged into a synchronized map being used by all threads in this task to save the processed values as they are being processed, since commits from the same project, user, and day, could be processed at the same time.

That synchronized map saved all metrics that are to be saved in the database. This map will be used for that purpose in the next tasklet.

Commits are processed one by one by each thread in a parallel way until one of the threads does not have an item to retrieve from the commit queue. When that happens, that thread sends a finish message to the other threads using that same queue. When every thread has finished, the next tasklet is executed.

MetricsWriterTasklet

This tasklet's purpose is to save the information processed into the database, and into log files to be used by the ELK stack.

Synchronized maps used in the tasklets beforehand are now used to retrieve the information needed to be saved.

This tasklet accesses CORQ's service implementation to save the metrics into the database if they are new, and change the metrics if they already exist in the database. For a metric to "already exist in the database", means that a metric with the same day, project, and user exists. Those are the unique attributes that define a metric.

The same is said about processed commits, and a processed commit is unique if the project and branch are the same, meaning we save the most recent processed commit for every branch in every project.

We also save information about all the new metrics generated into a log file that is to be used by Filebeats. This file is sent from Filebeats to the ELK stack at the server machine so that we can visualize the new information generated in Kibana at https://corq.opensoft.pt.

5.5.4 Index Reset Job

The purpose of this job is to extract all metric information from the database and create a log file containing that information in a format that is needed to be processed by the ELK stack.

To do so, the index reset tasklet accesses CORQ's service implementation to access information from the database about all the metrics.

A log file is created and one line is written for each metric object obtained from the database.

5.5.5 Local Projects Job

The purpose of this job is to populate projects that exist in the local machine of the user or administrator executing this job into the database.

To do so, the path on which these projects exist must be set in the appropriate property in the corq.local.properties file, and each folder must be named with the project key corresponding to the project inside the folder. This folder must also contain its .git folder.

This job will populate all projects in this directory into the database. Each project's key will be the folder's name. This key is very important so that there are no duplicate projects in the database. This can cause the metrics of a certain project to be generated twice.

5.5.6 Populate Database Job

The purpose of this job is to populate CORQ's database with Opensoft's collaborators' information, and with Opensoft's projects that exist in Bitbucket.

This job accesses CORQ's service implementation to manipulate the database.

UsersTasklet

This tasklet accesses CORQ Batch's service implementation to access Crowd's API and get information about the users in the Crowd platform. Crowd users have information about the users' e-mail, username, and display name. These attributes are used to create each user in the database. If the user exists in the database, it is not created again.

ProjectsTasklet

This tasklet accesses CORQ Batch's service implementation to access Bitbucket's API and get information about the project on the Bitbucket platform. Bitbucket projects have information about the projects' names and bitbucket keys.

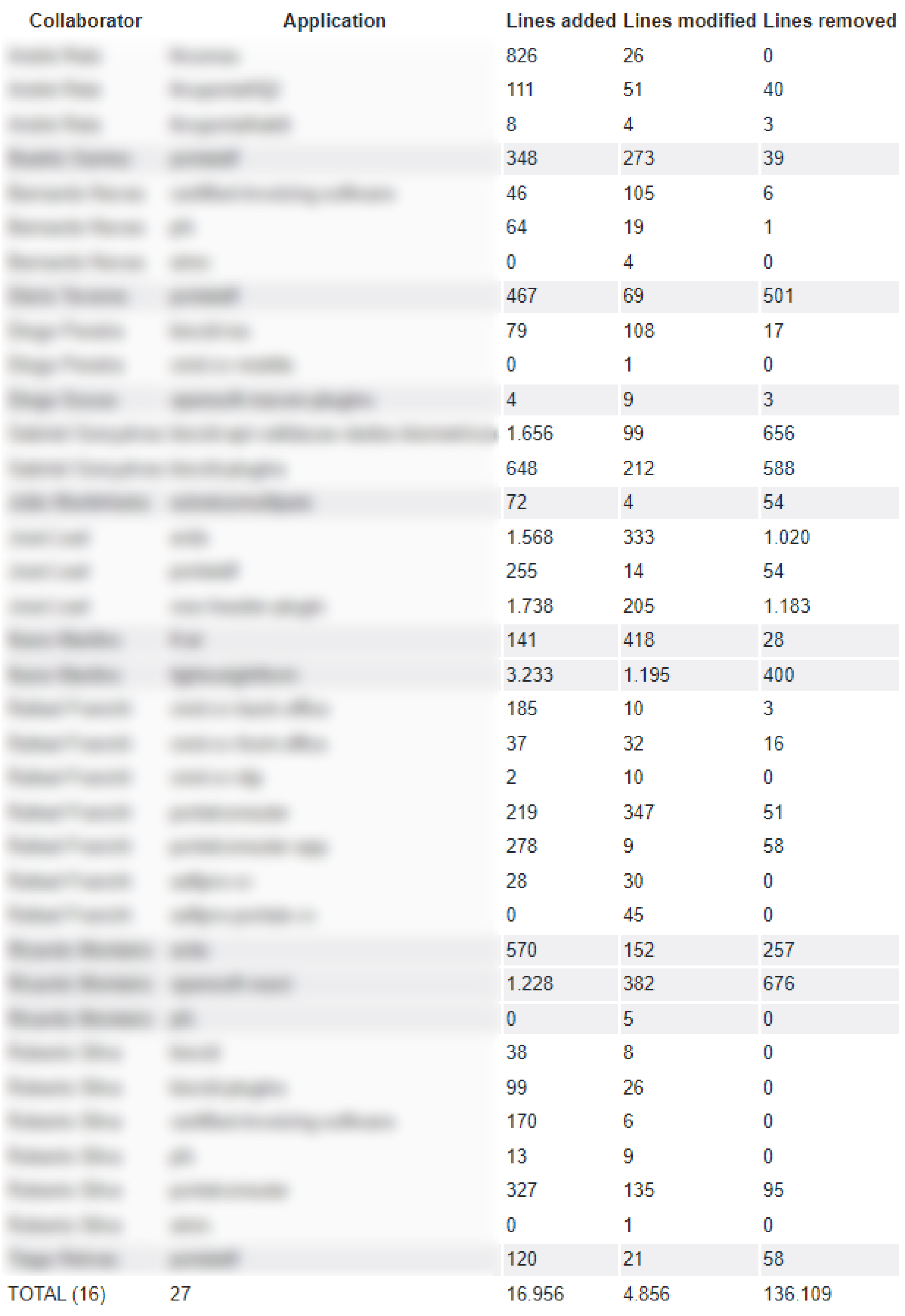
The other attributes are created by accessing the ”pom.xml” file, analyzing it, and processing it so that the other attributes needed such as the SonarQube key and Jira key are created. These are not used for now since they were not needed at this time of implementation and would require all projects at Opensoft to change their ”pom.xml” file. These attributes are used to create each project in the database. If the project already exists in the database, it is not created again.

5.5.7 Send E-mail Job

The purpose of this job is to send an e-mail with aggregated and organized information about metrics in an interval of time.

The job receives as input the start and end times of the interval from which to aggregate metric information in a yyyy-MM-dd (year-month-day) format.

The metrics are aggregated by application and collaborator. An example of an e-mail sent is shown in Figure 5.1. Some values are blurred for confidentiality reasons.



Collaborator	Application	Lines added	Lines modified	Lines removed
[redacted]	[redacted]	826	26	0
[redacted]	[redacted]	111	51	40
[redacted]	[redacted]	8	4	3
[redacted]	[redacted]	348	273	39
[redacted]	[redacted]	46	105	6
[redacted]	[redacted]	64	19	1
[redacted]	[redacted]	0	4	0
[redacted]	[redacted]	467	69	501
[redacted]	[redacted]	79	108	17
[redacted]	[redacted]	0	1	0
[redacted]	[redacted]	4	9	3
[redacted]	[redacted]	1.656	99	656
[redacted]	[redacted]	648	212	588
[redacted]	[redacted]	72	4	54
[redacted]	[redacted]	1.568	333	1.020
[redacted]	[redacted]	255	14	54
[redacted]	[redacted]	1.738	205	1.183
[redacted]	[redacted]	141	418	28
[redacted]	[redacted]	3.233	1.195	400
[redacted]	[redacted]	185	10	3
[redacted]	[redacted]	37	32	16
[redacted]	[redacted]	2	10	0
[redacted]	[redacted]	219	347	51
[redacted]	[redacted]	278	9	58
[redacted]	[redacted]	28	30	0
[redacted]	[redacted]	0	45	0
[redacted]	[redacted]	570	152	257
[redacted]	[redacted]	1.228	382	676
[redacted]	[redacted]	0	5	0
[redacted]	[redacted]	38	8	0
[redacted]	[redacted]	99	26	0
[redacted]	[redacted]	170	6	0
[redacted]	[redacted]	13	9	0
[redacted]	[redacted]	327	135	95
[redacted]	[redacted]	0	1	0
[redacted]	[redacted]	120	21	58
TOTAL (16)	27	16.956	4.856	136.109

Figure 5.1: E-mail example

This e-mail has information about the metrics generated in a month interval, on applications and collaborators of the company, and in this case about the projects in the company's Bitbucket platform.

5.6 Elastic search stack (ELK)

To ease the front-end development, and having into consideration the time needed and the time available to implement the front end, an outside solution was used to show the data generated by CORQ.

Elasticsearch is a distributed, free, and open search and analytics engine for all types of data. In our case, we use log files to structure the data generated by CORQ, and the ELK stack will read, analyze, and structure our data in a way that we have ways to analyze and visualize our data with Kibana.

Elasticsearch has many sub-components and in CORQ we use three of them, which are Kibana, Logstash, and Filebeats.

Each one serves its purpose, but they need to be configured before using in the server machine. For end-user usage, there is no need to configure anything.

The description of what each component does and how they interact is shown in Figure 5.2.

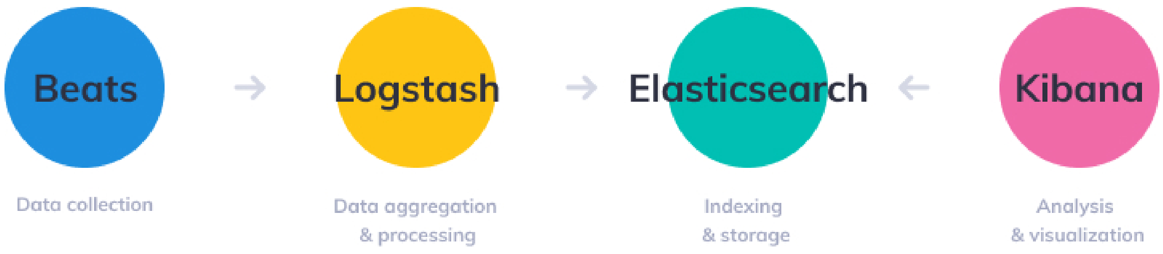


Figure 5.2: ELK Stack

Filebeats (Beats) collects data from the log files generated when generating metrics with CORQ, they are sent to the server machine through an ssh tunnel to Logstash at the server machine. That data is aggregated and processed into an index at Elasticsearch. Kibana will get the information on indexes existing in Elasticsearch and lets the end-user analyze and visualize it.

An *index* represents something that contains data that was processed and aggregated. In our case, the index's name is ”corq”.

Each graph, or view, is called a *visualization*. Each visualization can be edited or created by users with the right to do so. Each dashboard can contain one or more visualizations.

Some examples of graphs generated by Kibana with CORQ's data as shown in Figures 5.3, 5.4, 5.5 and 5.6. Some data in these Figures is blurred for confidentiality reasons.

In Figure 5.3 we have an example of CORQ's Kibana dashboard in the view of a user that can edit the dashboard. Although there are a lot of features to explore on a Kibana dashboard, the most important in our context is the following, wherein every dashboard we can as admins or users that has the right to do so:

- Create new visualizations.
- Edit existing visualizations.

- See existing visualizations in a specific time interval (that can be changed in the top right section where it is blurred, a normal user can too).
- Select different time intervals for different visualizations (normal users can too).

On Figure 5.4 we have the view of an admin or user that has the rights to edit a visualization, where we can most importantly:

- Change the data set we want to visualize the data from (this usually does not apply to CORQ since CORQ has one index associated with all the data available).
- Select what kind of view we want, such as line graph, vertical bar graph, horizontal bar graph, etc.
- The available fields with which we want to organize our graph.
- Select from the available fields which ones to set a horizontal axis or vertical axis, and what data to break down those axes from. Examples will be explained further in this section.
- Filter the axes and break down fields.

On Figures 5.5 and 5.6 we have two examples of visualizations already created. These two visualizations show CORQ's aggregated data in a time interval of two weeks. The horizontal axis represents the date field on a date per day way, meaning every section of this graph horizontally represents one day. On the vertical axis, we have the sum of value (value representing a data entry in the metric data of CORQ), and this value is filtered in a way that the metric_enum value associated with this sum of value is "LINES_ADDED". So we have the value of lines of code added since this value is filtered, per day, for two weeks.

This value is then broken down by collaborator or application, so we do not get the value of lines added for the whole company per day but divided by collaborator or application, depending on which we want to see. If we wanted to see the whole company we could too, and see the evolution of lines of code added to the company.

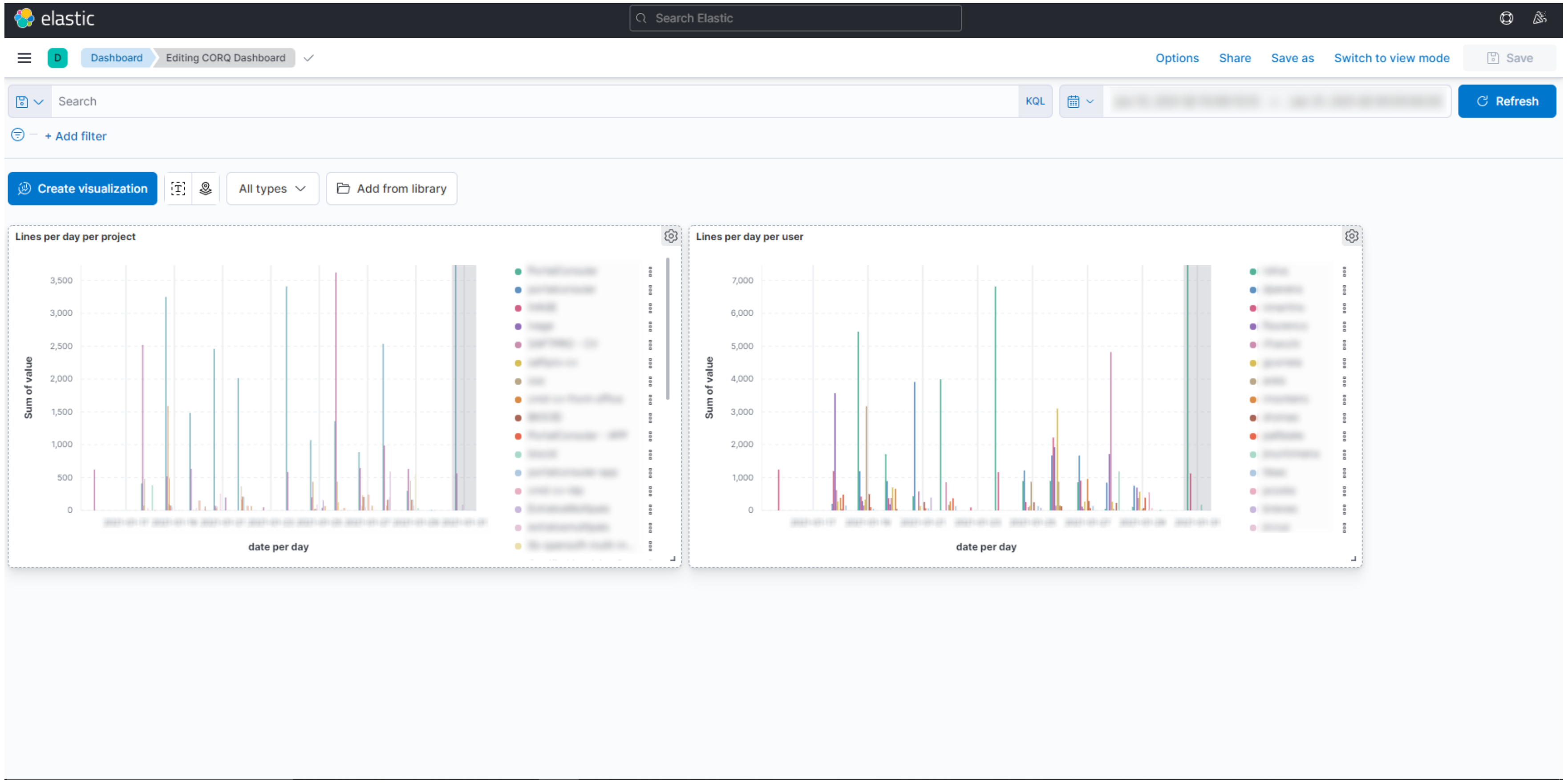
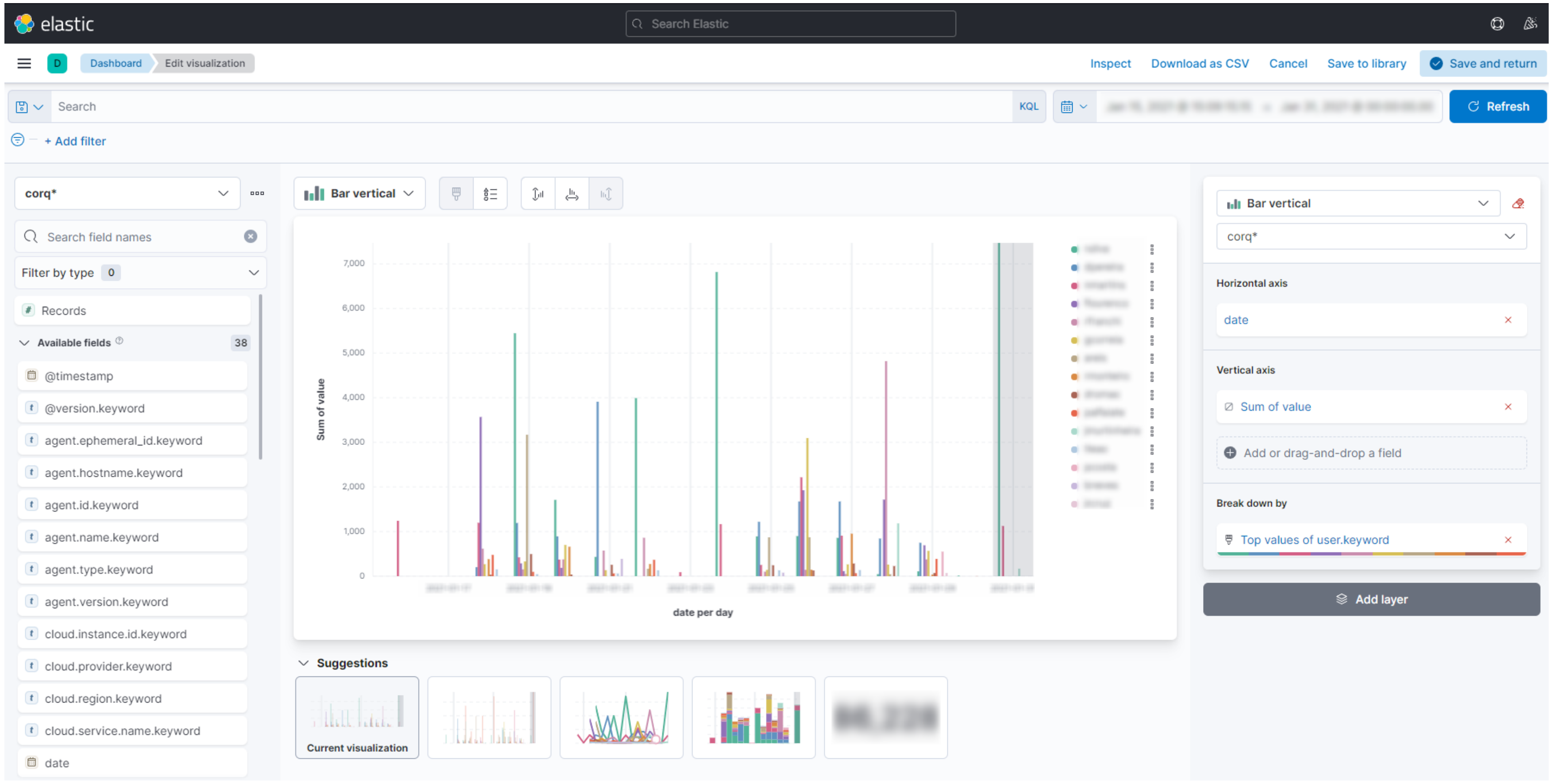


Figure 5.3: CORQ's Kibana dashboard



Horizontal axis: date

Vertical axis: Sum of value

Break down by: Top values of user.keyword

Figure 5.4: CORQ's Kibana visualization edit example

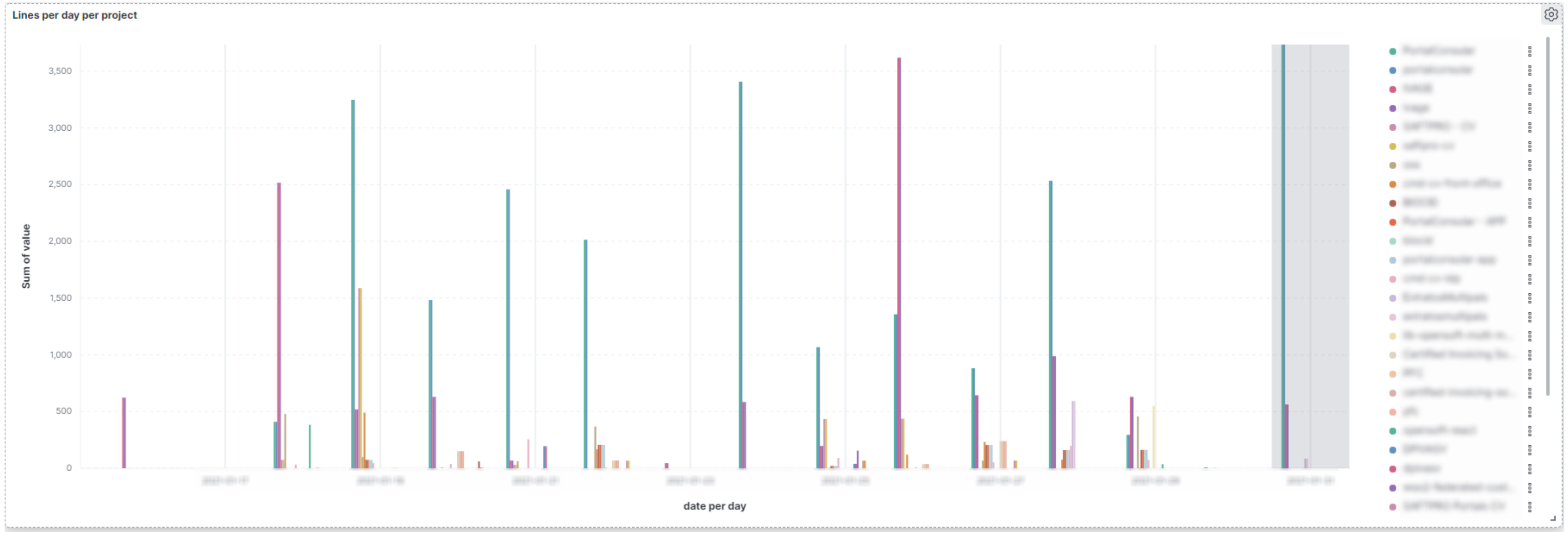


Figure 5.5: CORQ's lines per application

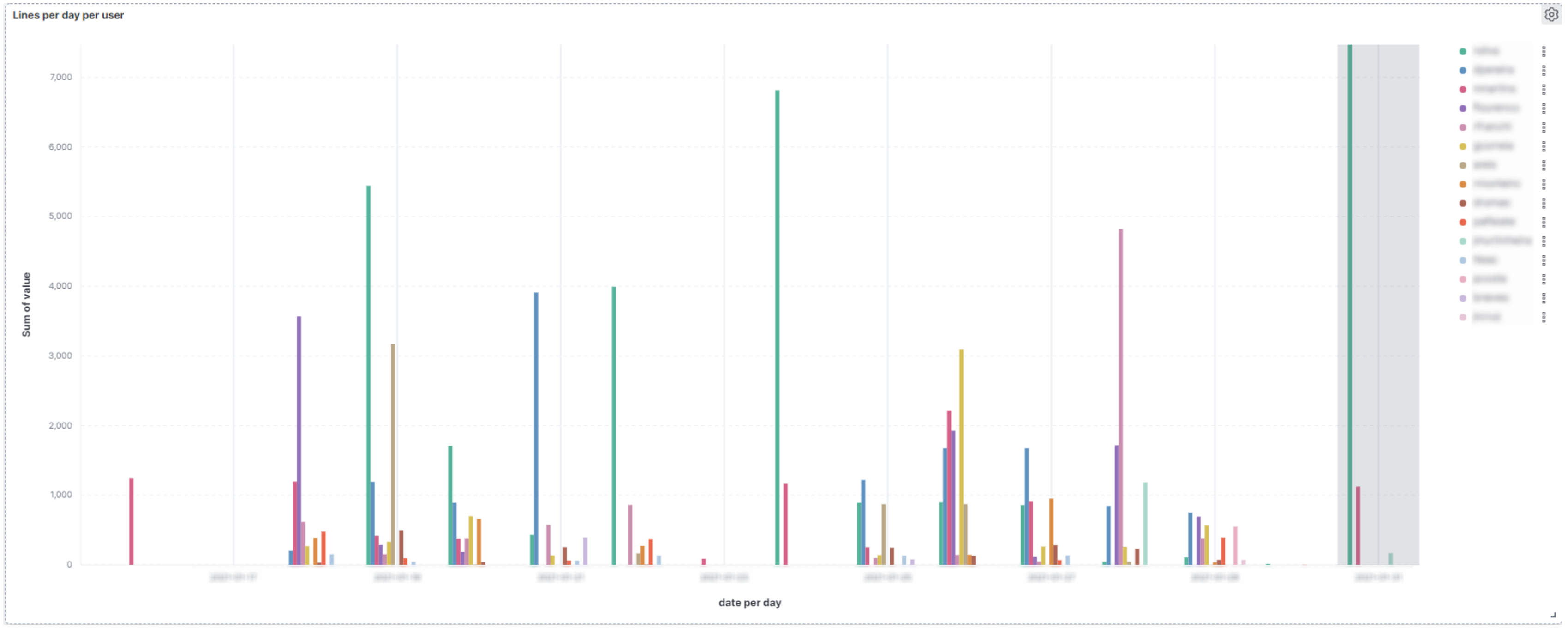


Figure 5.6: CORQ's lines per collaborator

5.7 Tests

The test plan was not documented because of the time resources available and the priorities, but the implementation of tests did start.

To implement tests in CORQ, JUnit version 4.13, Spring boot test version 4.3.6, and an H2 in-memory database were used. The test H2 in-memory database used the same entities and repositories defined with JPA in the Java code as the production and development context. When doing tests with the in-memory database, if there is a need to, data will be created before the tests and cleaned up after the tests.

It is very important that the test context replicate a real-life context of usage of the application, so the results of the tests would be the same as if the application was being run in a production context.

This way, when the application is deployed into production, when the tests are run with no errors, then when users use the application the risk of the application having errors is lower.

The implementation of what I would consider system tests for CORQ Batch jobs started but was not implemented successfully.

Tests were successfully implemented for entities (unit and integration tests) and repositories (integration tests) and the CORQ Batch and CORQ Common modules services (integration tests).

The logic behind the priorities regarding the project's tests was:

1. Start with tests on functions of the CORQ Common service implementation since if the access and manipulation to the database are not working, nothing else works. It makes sense that the most important module to start testing on is the commonly used for all other modules, even though right now CORQ Common only complements one module that being CORQ Batch, it will be the one to complement other new modules in the future.

2. Implement unit and integration tests for CORQ Batch service implementation.

3. Implement system tests for every batch job implemented in the CORQ Batch module. This is the harder-to-write type of test in this project, but very important to be as close to the user perspective when executing the CORQ Batch jobs (if they have to), since each job is executed independently.

4. Implement system or manual tests that include the ELK stack to ensure it is well configured and working. Both these tests and the CORQ Batch jobs system tests might become flaky, meaning not always the result will be the same with the same configuration since both CORQ Batch and CORQ Common modules communicate with a lot of outside interfaces, and depend on them.

The tests that were successfully implemented followed white-box testing techniques such as control and data flow coverage. These techniques were mainly used for the unit tests, and for tests that were not successfully implemented, since the integration tests implemented did not have any control or data flow coverage to consider since they mainly focused on database and outside interfaces access and manipulation, and tests on data structures used. The tests successfully implemented were:

- CORQ Common
  - Entity classes (Unit tests)
  - Repository classes (Integration tests)
  - Service implementation (Integration tests - Database Query related)
- CORQ Batch
  - Service implementation (Integration test - Bitbucket)

5.8 Deployment and artifacts produced

The deployment of CORQ is configured using Jenkins. Jenkins uses the configurations provided by Maven and pom.xml to have information on how the project is supposed to be built and what artifacts it is supposed to produce and deploy to the server machine. When pom.xml is configured, CORQ is one click away from being built and deployed to the server machine, or from a user to download the zip.

This zip file produced contains a folder with the following structure:

- **bin/** - contains binary files, which are executable files that the user or server executes to accomplish a certain purpose. The binary files produced will be described below.
- **config/** - contains .properties file that is used by any binary executable file to access certain properties defined by the user.
- **filebeats/** - folder containing all files needed by Filebeats, these files are used by some binary files to publish produced artifacts to the server by using an SSH tunnel.
- **lib/** - contains .jar files of dependencies used by corq-batch.
- **repositories/** - empty folder to be used in case the user decides to clone projects from the Bitbucket platform into the local machine, user, or server and decides to use this folder.
- **corq-batch.jar** - executable file of the corq-batch module.

The binary files produced for corq-batch are batch jobs that were divided into different executable files, and they are:

- **clone-repos** (.sh and .cmd) - Clone repositories from Opensoft's Bitbucket to the local machine in the path indicated in the file .properties, in the property "repositories.clone.path".
- **generate-metrics** (.sh and .cmd) - Generate metrics from the information existing in the repositories that are in the folder indicated in the property "repositories.clone.path", writing the metrics generated in the database and to a .log file that will be sent by Filebeats to the ELK stack.
- **index-reset** (.sh and .cmd) - Creates a .log file to be used by Filebeats, containing all information existing in the database about metrics. This executable is used whenever there are issues with the data existing in the ELK stack.
- **local-projects** (.sh and .cmd) - Creates all the projects that exist in the folder indicated in the property "repositories.clone.path" to the database.
- **populate-database** (.sh and .cmd) - Populates the database with users and projects. The users are the ones found at the Crowd API and the projects are imported from the Bitbucket API.
- **send-email** (.sh and .cmd) - Sends an e-mail with information about the metrics of a certain time interval. The start and end times are input by the user on the terminal when the executable is run.
- **open-database-ssh-tunnel**.cmd - Opens an ssh tunnel that is used with the purpose of accessing the database at the CORQ server machine.
- **corq-publish-to-filebeats**.cmd - Publishes the information of metrics generated, which are saved into a .log file, with Filebeats to the ELK stack at the server machine. To do this, an ssh tunnel is opened to the port at which Logstash is hosted in the CORQ server machine.
- **setDomainEnv**.cmd - Helper cmd file that sets environment variables for the execution of the executables described above, such as Java home, the CORQ jar location, CORQ's base directory, and CORQ's application.properties file.
- **springBatchProcessor**.cmd - Helper cmd file to execute all other cmd's above after setting the environment variables. Also sets the default memory usage needed.

5.9 Installation and usage guide

5.9.1 User

To use CORQ, the user must first download the zip artifact produced. This guide is based on a Windows environment since it is the operating system used by collaborators at Opensoft.

Before running CORQ's executables the user must have these available in their machine:

- Jdk11 or a higher version of JDK.
- Windows PowerShell.
- Windows cmd.

After that, the user will have to follow the steps described below:

1. If the user does not have the ssh key authorized to CORQ's server machine, then run the file inside the bin folder called **keygen-ssh-tunnel.ps1** with administrator rights.

   (a) This file will create an ssh key in the user's windows system so that an ssh tunnel to CORQ's server machine can be created.

   (b) If the user is having the following error:

   ```
   ..\keygen-ssh-tunnel.ps1 cannot be loaded
      because running scripts is disabled on
      this system.
   ```

   then they can run the following line of code to permit the windows PowerShell to run the script:

   ```
   Set-ExecutionPolicy -ExecutionPolicy
      RemoteSigned
   ```

   (c) After that uses the default settings shown as input (clicking enter) until the line asking for a password appears. The rest of this step cannot be described for confidentiality reasons, but the user will input a password to set up the key to the server machine.

2. After having the key to CORQ's server machine the user can run the executables they want to generate metrics and publish them to the ELK stack so they can see them in graphs. The first step is to open an ssh tunnel to the CORQ production database by running the **open-database-ssh-tunnel.cmd** file and leaving that window open, and it can be minimized.

3. Modify the properties in the file ”corq.local.properties” inside the ”config” folder. The properties to be changed are shown in a section containing ”# edit these” on the file so it is easier for the user to know. The properties are:

   (a) *verbose* - shows more output text for debug purposes.

   (b) *git.username* - the user’s git username so CORQ can have permission to access any git project that the user has on his local machine.

   (c) *git.password* - the user’s git password so CORQ can have permission to access any git project that the user has on his local machine.

   (d) *repositories.clone.path* - absolute path to the folder on which the user contains projects that he wishes to be added to CORQ’s database, in order to generate metrics about them.

      - **Note:** The name of the folder of each project must be the same as the project’s key in Bitbucket (or any other CVS platform that it exists).

4. If the user’s java home version is not jdk11 or higher, then the user must edit the file setDomainEnv.cmd that is in the folder bin, and edit the variable ”**JAVA_EXEC**” for the absolute path of ”java.exe” of the version 11 or higher.

5. Before generating metrics about projects existing in the user’s local machine they must exist in CORQ’s database. First, the user must put them in the same folder as the property ”repositories.clone.path” and change their names to the ones they have as keys in Bitbucket or any other CVS platform they exist.

   - **Note:** The following lines should not be alert as they show the normal functioning of the executable.

   ```
   No datasource was provided...using a Map
      based JobRepository
   No transaction manager was provided, using
      a ResourcelessTransactionManager
   ```

   - **Optional:** If the user wants to generate metrics of the projects in Bitbucket’s Opensoft instead of any project they have locally then the user can do so. To do so, run the file **clone-repos.sh**. This file will clone the projects directly from the Opensoft Bitbucket workspace to the folder defined in ”repositories.clone.path” by the user beforehand.

6. After having the projects in the database, we can generate the metrics by executing the file **generate-metrics.cmd**.

7. Verify if the log file is being used by Filebeats was created after the execution at filebeats\corq-logs folder

8. If the log file is in the folder, then the user can publish this log file to the **ELK stack** at the server machine. To do so, we have to run the file **corq-publish-to-filebeats.cmd** at the ”bin” folder.

9. After Filebeats has published the log file data to the ELK stack at the server machine, the metrics can be seen at **https://corq.opensoft.pt**. The credentials to enter this link are at a system that cannot be described for confidentiality reasons.

5.9.2 Server

To install the ELK stack (Latest version 7 available for all modules) and use CORQ in the server machine (Linux machine) and make CORQ usable for users in their machines we need to follow these steps:

1. Install Elasticsearch

   • Installation commands

   ```
   curl -fsSL https://artifacts.elastic.co/GPG-KEY-
      elasticsearch | sudo apt-key add -
   apt install elasticsearch=7.17.4
   ```

   • Comment the options ”xpack.security.enabled: true” e ”xpack.security.authc.api_key.enabled: true” in the file ”elasticsearch.yml” in the path ”/etc/elasticsearch”

2. Install Kibana

   ```
   apt install kibana=7.17.4

   #protect with basic auth: username + password from stdin
   echo "kibanaadmin:`openssl passwd -apr1`" | sudo tee -a
      /etc/apache2/.htpasswd
   ```

3. Install Logstash

   • Run the installing command

   ```
   apt install logstash=7.17.4
   ```

   • Create a configuration file in the path ”/etc/logstash/conf.d” with the following code:

   ```
   input {
      beats {
         port => 5044
   ```

```
      }
}

filter {
   dissect {
      mapping => {"message" => "%{metric_enum} ;
         %{origin} ; %{project} ; %{user} ;
         %{value} ; %{date}"}
   }
   mutate {
      convert => {
         "value" => "integer"
         "metric_enum" => "string"
         "origin" => "string"
         "project" => "string"
         "user" => "string"
      }
   }

  date {
      match => [ "date", "dd-MM-yyyy", "ISO8601" ]
  }
}

output {
  elasticsearch {
   hosts => ["http://localhost:9200"]
   index => "corq"
   document_id =>
      "%{metric_enum}-%{origin}-%{project}-%{user}
   -%{value}-%{date}"
  }
  stdout {
   codec => rubydebug
  }
}
```

4. Install Filebeats

- Run the installing command

```
apt install filebeats=7.17.4
```

- Edit options in the file ”filebeat.yml” in the path ”/etc/filebeat” as shown below

```
filebeat.inputs:

# Each - is an input. Most options can be set at
   the input level, so
```

```
# you can use different inputs for various
   configurations.
# Below are the input specific configurations.

# filestream is an input for collecting log
   messages from files.
- type: filestream

  # Change to true to enable this input
     configuration.
  enabled: true

  # Paths that should be crawled and fetched. Glob
     based paths.
  paths:
    - /home/corqx/corq-deployed/corq-batch-0.1.0/
       filebeats/corq-logs/*
# make the path above the following:
   {$CORQ_PATH}/filebeats/corq-logs/*
```

5. Enable and run services

   - Enable and run the services installed. Start with Elasticsearch and Kibana.

   ```
   service <service_name> enable
   service <service_name> start
   ```

   - After running Elasticsearch and Kibana, check in https://corq.opensoft.pt/ if both services are communicating and working as expected. The admin can also check this with the command

   ```
   service <service_name> status
   ```

   - Run the services Logstash and Filebeats.

6. (Optional) Run CORQ

   - After installing all the needed services, the admin can run CORQ on the server machine if they want to. Being in the machine there is no need to use SSH tunnels to access the database and to connect Filebeats to Logstash. Still, there are some configurations that we need to change before using CORQ.
     (a) *verbose* - shows more output text for debug purposes.
     (b) *git.username* - the user's git username so CORQ can have permission to access any git project that the user has on his local machine.
     (c) *git.password* - the user's git password so CORQ can have permission to access any git project that the user has on his local machine.

(d) *repositories.clone.path* - absolute path to the folder on which the user contains projects that he wishes to be added to CORQ's database, to generate metrics about them.
   - **Note:** The name of the folder of each project must be the same as the project's key in Bitbucket (or any other CVS platform that it exists).

(e) Use port 3306 for the database connection.

- After everything is running and configured the admin can run any executable in the folder bin.

Chapter 6

Conclusion

To conclude this project thesis, an evaluation of the work done, the difficulties in the whole process of this project, the acquired skills, and future work to be done on the project will be described and discussed.

6.1 Contributions

As a result of this thesis, the implementation of CORQ as a final product can be summarized as a reliable and extensible solution implemented by using Spring as the back-end, with a MariaDB (MySQL) database, that uses ELK Stack as aggregator and visualizer of the data generated by the solution with Spring Batch. Spring batch will generate metrics by accessing and manipulating data that is retrieved from platforms used at Opensoft such as Bitbucket, by accessing the platforms' APIs. My contributions to this implementation were the following:

- Exploring APIs of the platforms used to generate quality metrics, both implemented (Bitbucket) and not yet implemented (SonarQube, Jira) and documenting information on them.
- Analyzing and designing CORQ, by documenting all the artifacts that had been considered essential to the project's success, and that followed Opensoft's methodology of work, registering it in the Confluence platform.
- Implementation of CORQ using Java (Spring Boot and Batch and other complementary frameworks) and Maven (including the deployment of CORQ directly to the server machine), with the help of platforms such as SonarQube for code analysis and Bitbucket as a host for repositories, following the planning done at the start of the project, registering work time in the Jira platform.
- Exploring, configuring, installing, and integrating the ELK Stack into CORQ.

These contributions resulted in the project being fully documented, partly tested, and with a reliable and extensible back-end and ELK Stack implemented, with the metrics associated with lines of code (LOC) added, changed, and removed for repositories in local collaborators' machines, in the server machine, or projects that exist in the Opensoft Bitbucket platform.

In terms of the logic behind the implementation and documentation, all the work has been supervised and guided by my supervisor Luís Pereira and Ricardo Mascarenhas, a software architect at Opensoft. That includes code review and planning of the work that is needed to be done on a twice-a-week basis.

6.2 Evaluation

My work at Opensoft in the ten months developing CORQ has been a big accomplishment. Although not all the work did occur to plan, the results of my work with the guidance and also work of my supervisor Luis Pereira and Ricardo Mascarenhas, a software architect at Opensoft has shown to be positive and with a bright future. At the end of implementation, at the moment the ELK stack got implemented, and CORQ's data generated finally showed to be something that can be analyzed and taken results and conclusions from, was when the realization that the tool really could be used in a positive manner at the company.

Since CORQ can clone and analyze all projects in the Opensoft Bitbucket platform, we can see the evolution of lines of code on these repositories for the last 20 years. Such evolution can be interesting to analyze and make conclusions from. Even in a month, we can analyze in which period of a month or week the company is making the most adding, changing, or removing lines of code. If the collaborators produce fewer lines of code at the start of the week, maybe focus the start of the week with fewer code-intensive tasks, and focus on other tasks. Any metric or indicator of quality or productivity can have results that are not direct and need some extra analyzing and interpretation to get the results and conclusions needed without any prejudice of using these metrics. A lot of results and conclusions can be taken from analyzing a quantitative metric, then with the future metrics that could be implemented in CORQ, I am confident it could bring excellent results to Opensoft.

6.3 Difficulties

The main difficulties found while developing CORQ were the adaptation to everything at Opensoft. That includes the work times, since I worked on CORQ from the start of September until the last day of June for 40 hours a week, except for national and regional holidays. This is a full-time job working schedule and was difficult to adapt to especially at the start. Balancing the time to work on the project and write this thesis and the preliminary report was also a challenge.

Adapting to the methodology of work and platforms used at Opensoft also required some weeks of studying and practicing. Also, to set up the *software* used at the company, in the work computer provided by the company, such as IDE, VCS, main languages used, learning Spring, and others.

Another big difficulty was sticking to the planning. Some phases of implementation took more time than initially planned, and although not everything that was planned was implemented, the project has good extensibility and the team showed great adaptation to changes, as shown in the front-end situation. The main focus of the team at the end of the project was to have at least something that we can test, use, and experiment with, to have some results on how the project would be used in a real situation, and if it was good enough to keep working on it, with minimal metrics shown, and that was the case of CORQ at the end of implementation, and that was considered to be fulfilled by both the team that conceived and guided this project, and by Opensoft.

6.4 Acquired skills

The development of this project and thesis at Opensoft was a big accomplishment and a great experience that helped me grow as a person and as a software developer.

Skills such as programming with Java, Spring, Maven, and some workplace *software* and methodologies such as time registering, documentation platform, company code repositories, code analysis, and deployment of projects were learned and improved.

In my opinion, the biggest growth that I can take from this project and thesis was to manage the time available and use the time available the best way possible. This project had a lot of different features to be developed, and depending on the approach taken, could have come out very different. Prioritizing certain features so that at the end of the time available there is an extensible and usable project although not everything initially planned is implemented is very important so that we have palpable results, especially when the context of the project is one of a master thesis.

6.5 Future work

For future work, first of all, we have the features that were planned but the time available to have palpable results was not enough. These features, ordered by priority, were:

- Finishing integration and unit tests for each of the batch jobs implemented - this work was started but was not considered of high priority at the time of implementation.

- Add new metrics related to the APIs already implemented - with the time available for implementation, although the implementation for API access to Opensoft's SonarQube and Jira platforms were implemented, there was not enough time available to add metrics related to these platforms. More indicators implemented means more time CORQ saves the company, since reviewing these platforms can use a lot of time as explained in the analysis section of this report.
- Add new metrics related to other platforms such as Confluence for the same reasons explained above.
- Make CORQ run automatically at the host machine at a specific time every day (or in another customizable time span).

Although these were the features already thought out by me, and the team that was guiding me in the development of CORQ, since CORQ is considered an important tool for Opensoft's growth, there is a high probability that more features that were not already thought about will be implemented in the future.

The extensibility of the application implemented gives me enough confidence to say that any new feature thought to be implemented in CORQ can be implemented without excessive effort.

Bibliography

[1] Ermiyas Birihanu Belachew, Feidu Akmel Gobena, and Shumet Tadesse Nigatu. “Analysis of software quality using software metrics”. In: *International Journal on Computational Science & Applications (IJCSA)* 8.4/5 (2018).

[2] L. K. Campbell. “Software Metrics: Adding Engineering Rigor to a Currently Ephemeral Process”. In: (1995).

[3] Tsvetelina Mladenova. “Software Quality Metrics – Research, Analysis and Recommendation”. In: *2020 International Conference Automatics and Informatics (ICAI)*. 2020, pp. 1–5. DOI: 10.1109/ICAI50593.2020.9311361.

[4] George K Thiruvathukal, Nicholas J Hayward, Konstantin Läufer, et al. “Metrics dashboard: A hosted platform for software quality metrics”. In: *arXiv preprint arXiv:1804.02053* (2018).

[5] Luc Filion et al. “Using Atlassian tools for efficient requirements management: An industrial case study”. In: *2017 Annual IEEE International Systems Conference (SysCon)*. 2017, pp. 1–6. DOI: 10.1109/SYSCON.2017.7934769.

[6] Wang Yiran, Zhang Tongyang, and Guo Yidong. “Design and implementation of continuous integration scheme based on Jenkins and Ansible”. In: *2018 International Conference on Artificial Intelligence and Big Data (ICAIBD)*. 2018, pp. 245–249. DOI: 10.1109/ICAIBD.2018.8396203.

[7] Sung Jun Son and Youngmi Kwon. “Performance of ELK stack and commercial system in security log analysis”. In: *2017 IEEE 13th Malaysia International Conference on Communications (MICC)*. 2017, pp. 187–190. DOI: 10.1109/MICC.2017.8311756.

[8] Raimund Moser, Witold Pedrycz, and Giancarlo Succi. “A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction”. In: *2008 ACM/IEEE 30th International Conference on Software Engineering*. 2008, pp. 181–190. DOI: 10.1145/1368088.1368114.

[9] Redhat. *REST vs. SOAP*. URL: https://www.redhat.com/en/topics/integration/whats-the-difference-between-soap-rest. accessed: 29.09.2022).

[10] Anna Monus. *SOAP vs REST vs JSON - a 2021 comparison*. URL: https://raygun.com/blog/soap-vs-rest-vs-json/. accessed: 29.09.2022).

[11] Matthew Gilliard. *5 ways to make HTTP requests in Java*. URL: https://www.twilio.com/blog/5-ways-to-make-http-requests-in-java. accessed: 29.09.2022).

[12] Baeldung. *Performance of Java Mapping Frameworks*. URL: https://www.baeldung.com/java-performance-mapping-frameworks. accessed: 29.09.2022).

[13] Drazen Nikolic. *Spring WebClient vs. RestTemplate*. URL: https://www.baeldung.com/spring-webclient-resttemplate. accessed: 29.09.2022).